

# Microsoft Excel Object Model

## Application

- AddIns |
  - AddIn
- AutoRecover
- CellFormat
  - Borders
    - Border
  - Font
  - Interior
- DefaultWebOptions
- Dialogs
  - Dialog
- ErrorCheckingOptions
- Names
- ODBCErrors
- OLEDBErrors
- Range
  - Areas
  - Borders
    - Border
  - Characters
  - Comment
    - Shape
  - Errors
    - Error
  - Font

## Range (continued)

- ListObject
  - ListColumns
  - ListRows
  - XmlMap
- Phonetic
- Phonetics
- PivotCell
  - PivotItemList
- PivotField
  - CubeField
- PivotItem
- PivotTable
  - CalculatedMembers
  - CubeFields
  - PivotFormulas
- QueryTable
  - Parameters
- SoundNote
- Validation
- Worksheet
  - AutoFilter
  - Comments
  - CustomProperties
  - HPageBreaks

- RecentFiles
  - RecentFile
- RTD
- Sheets
  - HPageBreaks
    - HPageBreak
  - VPageBreaks
    - VPageBreak
- SmartTagRecogn
  - SmartTagReco
- Speech
- SpellingOptions
- UsedObjects
- Watches
  - Watch
- Windows
  - Window
    - Panes
- Workbook
  - CustomViews
  - Mailer
  - PublishObjects
    - PublishObje
  - RoutingSlip

- [FormatConditions](#)
- [Hyperlinks](#)
- [Interior](#)

### Legend

Object and collection  
Object only

- [ListObjects](#)
- [Outline](#)
- [PageSetup](#)
- [Protection](#)
- [QueryTables](#)
- [Shapes](#)
- [Tab](#)
- [VPageBreaks](#)
- [XPath](#)
- [XmlMap](#)

- [SmartTagOptions](#)
- [Styles](#)
  - [Style](#)
- [WebOptions](#)
- [XmlMaps](#)
  - [XmlMap](#)
- [XmlNamespaces](#)
  - [XmlNamespaces](#)
- [Workbooks](#)
- [WorksheetFunctions](#)

Please refer to the following links for more information on notable Excel objects

[ChartObject object](#)

[Name object](#)

[Shapes collection](#)

[SmartTag object](#)

[Show All](#)

# New Objects

For the latest information about programming with Microsoft Excel, including product news, technical articles, downloads, and samples, visit the Microsoft Office Developer Center on the [Microsoft Developer Network \(MSDN\)](#) Web site.

The following table lists objects added to the Microsoft Office Excel 2003 object model.

<b>Object</b>	<b>Description</b>
<a href="#"><u>ListColumn</u></a>	Represents a column in a <a href="#">list</a> . The <b>ListColumn</b> object is a member of the <b>ListColumns</b> collection. The <b>ListColumns</b> collection contains all the columns in a list ( <b>ListObject</b> object).
<a href="#"><u>ListColumns</u></a>	A collection of all the <b>ListColumn</b> objects in the specified <b>ListObject</b> object. Each <b>ListColumn</b> object represents a column in the list.
<a href="#"><u>ListDataFormat</u></a>	The <b>ListDataFormat</b> object holds all the data type properties of the <b>ListColumn</b> object.
<a href="#"><u>ListObject</u></a>	Represents a list object on a worksheet. The <b>ListObject</b> object is a member of the <b>ListObjects</b> collection. The <b>ListObjects</b> collection contains all the list objects on a worksheet.
<a href="#"><u>ListObjects</u></a>	A collection of all the <b>ListObject</b> objects on a worksheet. Each <b>ListObject</b> object represents a list in the worksheet.
<a href="#"><u>ListRow</u></a>	Represents a row in a List object. The <b>ListRow</b> object is a member of the <b>ListRows</b> collection. The <b>ListRows</b> collection contains all the rows in a list object.
<a href="#"><u>ListRows</u></a>	A collection of all the <b>ListRow</b> objects in the specified <b>ListObject</b> object. Each <b>ListRow</b> object represents a row in the list.
<a href="#"><u>XmlDataBinding</u></a>	Represents the connection to the source data for an <b>XmlMap</b> object.
<a href="#"><u>XmlMap</u></a>	

Represents an XML map that has been added to a workbook.

**XmlMaps**

Represents the collection of **XmlMap** objects that have been added to a workbook.

**XmlNamespace**

Represents a namespace that has been added to a workbook.

**XmlNamespaces**

Represents the collection of **XmlNamespace** objects in a workbook.

**XmlSchema**

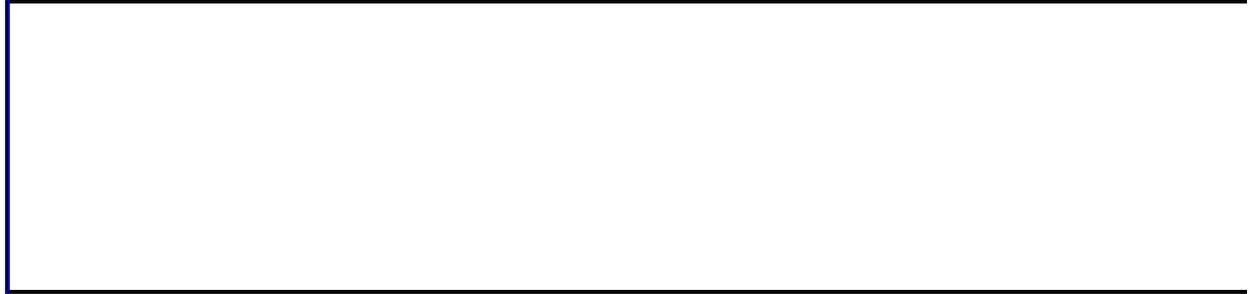
Represents an XML schema contained by an **XmlMap** object.

**XmlSchemas**

Represents the collection of **XmlSchema** objects contained by an **XmlMap** object.

**XPath**

Represents an XPath that has been mapped to a **Range** or **ListColumn** object.



# New Properties (Alphabetical List)

For the latest information about programming with Microsoft Excel, including product news, technical articles, downloads, and samples, visit the Microsoft Office Developer Center on the [Microsoft Developer Network \(MSDN\)](#) Web site.

The following table lists properties added to the Microsoft Office Excel 2003 object model (sorted alphabetically).

<b>New Property</b>	<b>Object(s)</b>
<a href="#"><b>Active</b></a>	ListObject
<a href="#"><b>ActiveXControl</b></a>	SmartTagAction
<a href="#"><b>AllowFillIn</b></a>	ListDataFormat
<a href="#"><b>AppendOnImport</b></a>	XmlMap
<a href="#"><b>ArbitraryXMLSupportAvailable</b></a>	Application
<a href="#"><b>AutoExpandListRange</b></a>	AutoCorrect
<a href="#"><b>CheckboxState</b></a>	SmartTagAction
<a href="#"><b>Choices</b></a>	ListDataFormat
<a href="#"><b>DataBinding</b></a>	XmlMap
<a href="#"><b>DecimalPlaces</b></a>	ListDataFormat
<a href="#"><b>DefaultValue</b></a>	ListDataFormat
<a href="#"><b>DisplayDocumentActionTaskPane</b></a>	Application
<a href="#"><b>DisplayInkComments</b></a>	Application, Workbook
<a href="#"><b>DocumentLibraryVersions</b></a>	Workbook
<a href="#"><b>ExpandHelp</b></a>	SmartTagAction
<a href="#"><b>HeaderRowRange</b></a>	ListObject
<a href="#"><b>InactiveListBorderVisible</b></a>	Workbook
<a href="#"><b>InsertRowRange</b></a>	ListObject
<a href="#"><b>InvalidData</b></a>	ListRow
<a href="#"><b>IsExportable</b></a>	XmlMap
<a href="#"><b>IsPercent</b></a>	ListDataFormat
<a href="#"><b>Icid</b></a>	ListDataFormat

<a href="#"><u>ListColumns</u></a>	ListObject
<a href="#"><u>ListDataFormat</u></a>	ListColumn
<a href="#"><u>ListDataValidation</u></a>	ErrorCheckingOptions
<a href="#"><u>ListObject</u></a>	QueryTable, Range
<a href="#"><u>ListObjects</u></a>	Worksheet
<a href="#"><u>ListRows</u></a>	ListObject
<a href="#"><u>ListSelection</u></a>	SmartTagAction
<a href="#"><u>Map</u></a>	XPath
<a href="#"><u>MaxCharacters</u></a>	ListDataFormat
<a href="#"><u>MaxNumber</u></a>	ListDataFormat
<a href="#"><u>MinNumber</u></a>	ListDataFormat
<a href="#"><u>Namespace</u></a>	XmlSchema
<a href="#"><u>Permission</u></a>	Workbook
<a href="#"><u>Prefix</u></a>	XmlNamespace
<a href="#"><u>PresentInPane</u></a>	SmartTagAction
<a href="#"><u>PreserveColumnFilter</u></a>	XmlMap
<a href="#"><u>PreserveNumberFormatting</u></a>	XmlMap
<a href="#"><u>RadioGroupSelection</u></a>	SmartTagAction
<a href="#"><u>Repeating</u></a>	XPath
<a href="#"><u>Required</u></a>	ListDataFormat
<a href="#"><u>RootElementName</u></a>	XmlMap
<a href="#"><u>RootElementNamespace</u></a>	XmlMap
<a href="#"><u>SaveDataSourceDefinition</u></a>	XmlMap
<a href="#"><u>Schemas</u></a>	XmlMap
<a href="#"><u>SharedWorkspace</u></a>	Workbook
<a href="#"><u>SharePointFormula</u></a>	ListColumn
<a href="#"><u>SharePointURL</u></a>	ListObject
<a href="#"><u>ShowAutoFilter</u></a>	ListObject
<a href="#"><u>ShowImportExportValidationErrors</u></a>	XmlMap
<a href="#"><u>ShowTotals</u></a>	ListObject
<a href="#"><u>SmartDocument</u></a>	Workbook
<a href="#"><u>SourceUrl</u></a>	XmlDataBinding
<a href="#"><u>Sync</u></a>	Workbook

[SyncScrollingSideBySide](#)

[TextboxText](#)

[TextFileVisualLayout](#)

[TotalsCalculation](#)

[TotalsRowRange](#)

[Uri](#)

[XmlMap](#)

[XmlMaps](#)

[XmlNamespaces](#)

[XPath](#)

**Windows**

**SmartTagAction**

**QueryTable**

**ListColumn**

**ListObject**

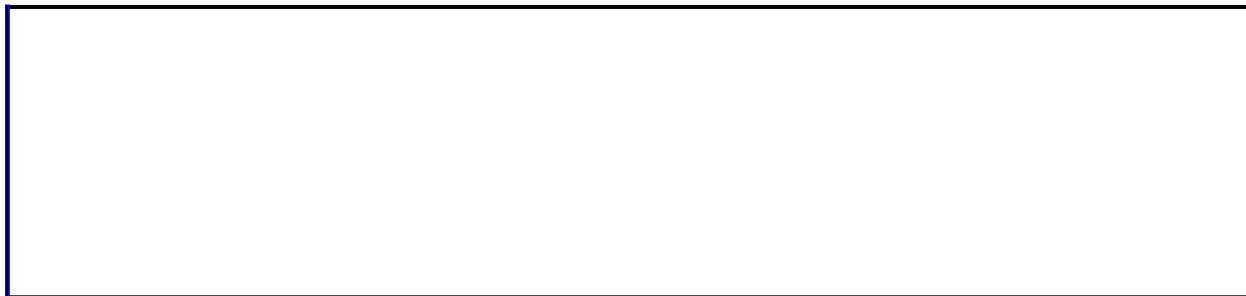
**XmlNamespace**

**ListObject**

**Workbook**

**Workbook**

**Range, ListColumn**



# New Properties (by Object)

For the latest information about programming with Microsoft Excel, including product news, technical articles, downloads, and samples, visit the Microsoft Office Developer Center on the [Microsoft Developer Network \(MSDN\)](#) Web site.

The following table lists properties added to the Microsoft Office Excel 2003 object model (sorted by object name).

<b>Object</b>	<b>New Properties</b>
<b>Application</b>	<a href="#">ArbitraryXMLSupportAvailable</a> , <a href="#">DisplayDocumentActionTaskPane</a>
<b>AutoCorrect</b>	<a href="#">AutoExpandListRange</a>
<b>ErrorCheckingOptions</b>	<a href="#">ListDataValidation</a>
<b>ListColumn</b>	<a href="#">ListDataFormat</a> , <a href="#">SharePointFormula</a> , <a href="#">TotalsCalculation</a> , <a href="#">XPath</a>
<b>ListDataFormat</b>	<a href="#">AllowFillIn</a> , <a href="#">Choices</a> , <a href="#">DecimalPlaces</a> , <a href="#">DefaultValue</a> , <a href="#">IsPercent</a> , <a href="#">Icid</a> , <a href="#">MaxCharacters</a> , <a href="#">MaxNumber</a> , <a href="#">MinNumber</a> , <a href="#">Required</a>
<b>ListObject</b>	<a href="#">Active</a> , <a href="#">HeaderRowRange</a> , <a href="#">InsertRowRange</a> , <a href="#">ListColumns</a> , <a href="#">ListRows</a> , <a href="#">SharePointURL</a> , <a href="#">ShowAutoFilter</a> , <a href="#">ShowTotals</a> , <a href="#">TotalsRowRange</a> , <a href="#">XmlMap</a>
<b>ListRow</b>	<a href="#">InvalidData</a>
<b>QueryTable</b>	<a href="#">ListObject</a> , <a href="#">TextFileVisualLayout</a>
<b>Range</b>	<a href="#">ListObject</a> , <a href="#">XPath</a>
<b>SmartTagAction</b>	<a href="#">ActiveXControl</a> , <a href="#">CheckboxState</a> , <a href="#">ExpandHelp</a> , <a href="#">ListSelection</a> , <a href="#">PresentInPane</a> , <a href="#">RadioGroupSelection</a> , <a href="#">TextboxText</a>
<b>Windows</b>	<a href="#">SyncScrollingSideBySide</a> <a href="#">DisplayInkComments</a> , <a href="#">DocumentLibraryVersions</a> , <a href="#">InactiveListBorderVisible</a> , <a href="#">Permission</a> ,
<b>Workbook</b>	<a href="#">SharedWorkspace</a> , <a href="#">SmartDocument</a> , <a href="#">Sync</a> ,

<b>Worksheet</b>	<a href="#"><u>XmlMaps</u></a> , <a href="#"><u>XmlNamespaces</u></a>
<b>XmlDataBinding</b>	<a href="#"><u>ListObjects</u></a> <a href="#"><u>SourceUrl</u></a> <a href="#"><u>AppendOnImport</u></a> , <a href="#"><u>DataBinding</u></a> , <a href="#"><u>IsExportable</u></a> , <a href="#"><u>PreserveColumnFilter</u></a> , <a href="#"><u>PreserveNumberFormatting</u></a> , <a href="#"><u>RootElementName</u></a> , <a href="#"><u>RootElementNamespace</u></a> , <a href="#"><u>SaveDataSourceDefinition</u></a> , <a href="#"><u>Schemas</u></a> , <a href="#"><u>ShowImportExportValidationErrors</u></a>
<b>XmlMap</b>	
<b>XmlNamespace</b>	<a href="#"><u>Prefix</u></a> , <a href="#"><u>Uri</u></a>
<b>XmlSchema</b>	<a href="#"><u>Namespace</u></a>
<b>XPath</b>	<a href="#"><u>Map</u></a> , <a href="#"><u>Repeating</u></a>



# New Methods (Alphabetical List)

For the latest information about programming with Microsoft Excel, including product news, technical articles, downloads, and samples, visit the Microsoft Office Developer Center on the [Microsoft Developer Network \(MSDN\)](#) Web site.

The following table lists methods added to the Microsoft Office Excel 2003 object model (sorted alphabetically).

<b>New Method</b>	<b>Object</b>
<a href="#"><u>BreakSideBySide</u></a>	Windows
<a href="#"><u>ClearSettings</u></a>	XmlDataBinding
<a href="#"><u>CompareSideBySideWith</u></a>	Windows
<a href="#"><u>DisplayXMLSourcePane</u></a>	Application
<a href="#"><u>ExportXml</u></a>	XmlMap
<a href="#"><u>ImportXml</u></a>	XmlMap
<a href="#"><u>InstallManifest</u></a>	XmlNamespaces
<a href="#"><u>LoadSettings</u></a>	XmlDataBinding
<a href="#"><u>ResetPositionsSideBySide</u></a>	Windows
<a href="#"><u>SaveAsXMLData</u></a>	Workbook
<a href="#"><u>SendFaxOverInternet</u></a>	Workbook
<a href="#"><u>SetValue</u></a>	XPath
<a href="#"><u>Unlink</u></a>	ListObject
<a href="#"><u>Unlist</u></a>	ListObject
<a href="#"><u>UpdateChanges</u></a>	ListObject
<a href="#"><u>XmlDataQuery</u></a>	Worksheet
<a href="#"><u>XmlImport</u></a>	Workbook
<a href="#"><u>XmlImportXml</u></a>	Workbook
<a href="#"><u>XmlMapQuery</u></a>	Worksheet



# New Methods (by Object)

For the latest information about programming with Microsoft Excel, including product news, technical articles, downloads, and samples, visit the Microsoft Office Developer Center on the [Microsoft Developer Network \(MSDN\)](#) Web site.

The following table lists methods added to the Microsoft Office Excel 2003 object model (sorted by object name).

New Method	Object
	<a href="#">DisplayXMLSourcePane</a>
ListObject	<a href="#">Unlink</a> , <a href="#">Unlist</a> , <a href="#">UpdateChanges</a>
Windows	<a href="#">BreakSideBySide</a> , <a href="#">CompareSideBySideWith</a> , <a href="#">ResetPositionsSideBySide</a>
Workbook	<a href="#">SaveAsXMLData</a> , <a href="#">SendFaxOverInternet</a> , <a href="#">XmlImport</a> , <a href="#">XmlImportXml</a>
Worksheet	<a href="#">XmlDataQuery</a> , <a href="#">XmlMapQuery</a>
XmlDataBinding	<a href="#">ClearSettings</a> , <a href="#">LoadSettings</a>
XmlMap	<a href="#">ExportXml</a> , <a href="#">ImportXml</a>
XmlNamespaces	<a href="#">InstallManifest</a>
XPath	<a href="#">SetValue</a>



# New Events

For the latest information about programming with Microsoft Excel, including product news, technical articles, downloads, and samples, visit the Microsoft Office Developer Center on the [Microsoft Developer Network \(MSDN\)](#) Web site.

The following table lists events added to the Microsoft Office Excel 2003 object model.

<b>New Event</b>	<b>Object</b>
<a href="#"><b><u>AfterXmlExport</u></b></a>	<b>Workbook</b>
<a href="#"><b><u>AfterXmlImport</u></b></a>	<b>Workbook</b>
<a href="#"><b><u>BeforeXmlExport</u></b></a>	<b>Workbook</b>
<a href="#"><b><u>BeforeXmlImport</u></b></a>	<b>Workbook</b>
<a href="#"><b><u>Sync</u></b></a>	<b>Workbook</b>
<a href="#"><b><u>WorkbookAfterXmlExport</u></b></a>	<b>Application</b>
<a href="#"><b><u>WorkbookAfterXmlImport</u></b></a>	<b>Application</b>
<a href="#"><b><u>WorkbookBeforeXmlExport</u></b></a>	<b>Application</b>
<a href="#"><b><u>WorkbookBeforeXmlImport</u></b></a>	<b>Application</b>
<a href="#"><b><u>WorkbookSync</u></b></a>	<b>Application</b>

---

# AddIns Collection Object

[Application](#) └ [AddIns](#)  
└ [AddIn](#)

A collection of [AddIn](#) objects that represents all the add-ins available to Microsoft Excel, regardless of whether they're installed. This list corresponds to the list of add-ins displayed in the **Add-Ins** dialog box (**Tools** menu).

## Using the AddIns Collection

Use the **AddIns** method to return the **AddIns** collection. The following example creates a list that contains the names and installed states of all the available add-ins.

```
Sub DisplayAddIns()  
    Worksheets("Sheet1").Activate  
    rw = 1  
    For Each ad In Application.AddIns  
        Worksheets("Sheet1").Cells(rw, 1) = ad.Name  
        Worksheets("Sheet1").Cells(rw, 2) = ad.Installed  
        rw = rw + 1  
    Next  
End Sub
```

Use the **Add** method to add an add-in to the list of available add-ins. The **Add** method adds an add-in to the list but doesn't install the add-in. Set the **Installed** property of the add-in to **True** to install the add-in. To install an add-in that doesn't appear in the list of available add-ins, you must first use the **Add** method and then set the **Installed** property. This can be done in a single step, as shown in the following example (note that you use the name of the add-in, not its title, with the **Add** method).

```
AddIns.Add("generic.xll").Installed = True
```

Use **AddIns(index)** where *index* is the add-in title or index number to return a single **AddIn** object. The following example installs the Analysis Toolpak add-in.

```
AddIns("analysis toolpak").Installed = True
```

Don't confuse the add-in title, which appears in the **Add-Ins** dialog box, with the add-in name, which is the file name of the add-in. You must spell the add-in title exactly as it's spelled in the **Add-Ins** dialog box, but the capitalization doesn't have to match.



# AllowEditRanges Collection

[Protection](#) └ [AllowEditRanges](#)

└ [AllowEditRange](#)

└ Multiple objects

A collection of all the [AllowEditRanges](#) objects that represent the cells that can be edited on a protected worksheet.

## Using the AllowEditRanges Collection

Use the [AllowEditRanges](#) property of the [Protection](#) object to return an **AllowEditRanges** collection.

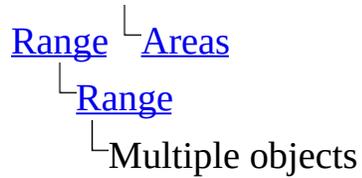
Once an **AllowEditRanges** collection has been returned, you can use the [Add](#) method to add a range that can be edited on a protected worksheet.

In this example, Microsoft Excel allows edits to range "A1:A4" on the active worksheet and notifies the user of the title and address of the specified range.

```
Sub UseAllowEditRanges()  
  
    Dim wksOne As Worksheet  
    Dim wksPassword As String  
  
    Set wksOne = Application.ActiveSheet  
  
    ' Unprotect worksheet.  
    wksOne.Unprotect  
  
    wksPassword = InputBox ("Enter password for the worksheet")  
  
    ' Establish a range that can allow edits  
    ' on the protected worksheet.  
    wksOne.Protection.AllowEditRanges.Add _  
        Title:="Classified", _  
        Range:=Range("A1:A4"), _  
        Password:=wksPassword  
  
    ' Notify the user  
    ' the title and address of the range.  
    With wksOne.Protection.AllowEditRanges.Item(1)  
        MsgBox "Title of range: " & .Title  
        MsgBox "Address of range: " & .Range.Address  
    End With  
  
End Sub
```



# Areas Collection



A collection of the areas, or contiguous blocks of cells, within a selection. There's no singular Area object; individual members of the **Areas** collection are **Range** objects. The **Areas** collection contains one **Range** object for each discrete, contiguous range of cells within the selection. If the selection contains only one area, the **Areas** collection contains a single **Range** object that corresponds to that selection.

## Using the Areas Collection

Use the **Areas** property to return the **Areas** collection. The following example clears the current selection if it contains more than one area.

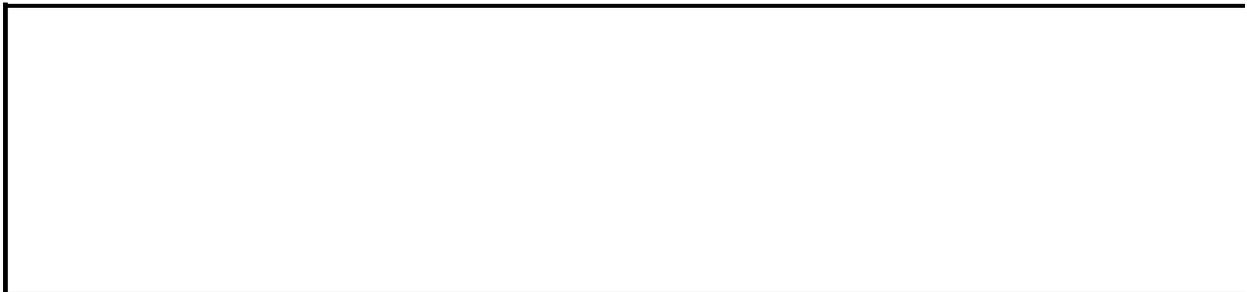
```
If Selection.Areas.Count <> 1 Then Selection.Clear
```

Use **Areas(index)**, where *index* is the area index number, to return a single **Range** object from the collection. The index numbers correspond to the order in which the areas were selected. The following example clears the first area in the current selection if the selection contains more than one area.

```
If Selection.Areas.Count <> 1 Then  
    Selection.Areas(1).Clear  
End If
```

Some operations cannot be performed on more than one area in a selection at the same time; you must loop through the individual areas in the selection and perform the operations on each area separately. The following example performs the operation named "myOperation" on the selected range if the selection contains only one area; if the selection contains multiple areas, the example performs myOperation on each individual area in the selection.

```
Set rangeToUse = Selection  
If rangeToUse.Areas.Count = 1 Then  
    myOperation rangeToUse  
Else  
    For Each singleArea in rangeToUse.Areas  
        myOperation singleArea  
    Next  
End If
```



# Axes Collection Object

[Axes](#)  [Axis](#)  
 Multiple objects

A collection of all the [Axis](#) objects in the specified chart.

## Using the Axes Collection

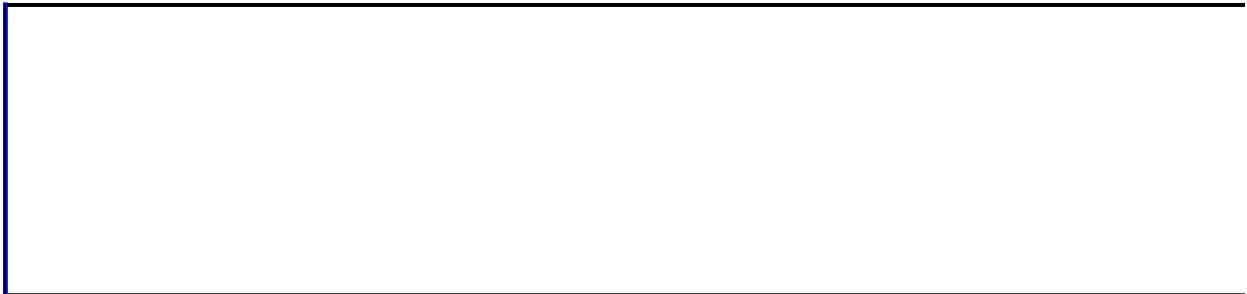
Use the **Axes** method to return the **Axes** collection. The following example displays the number of axes on embedded chart one on worksheet one.

```
With Worksheets(1).ChartObjects(1).Chart
    MsgBox .Axes.Count
End With
```

Use **Axes**(*type*, *group*), where *type* is the axis type and *group* is the axis group, to return a single **Axis** object. *Type* can be one of the following **XlAxisType** constants: **xlCategory**, **xlSeries**, or **xlValue**. *Group* can be one of the following **XlAxisGroup** constants: **xlPrimary** or **xlSecondary**. For more information, see the [Axes](#) method.

The following example sets the category axis title text on the chart sheet named "Chart1."

```
With Charts("chart1").Axes(xlCategory)
    .HasTitle = True
    .AxisTitle.Caption = "1994"
End With
```



# Borders Collection

Multiple objects [└ Borders](#)  
[└ Border](#)

A collection of four [Border](#) objects that represent the four borders of a [Range](#) or [Style](#) object.

## Using the Borders Collection

Use the **Borders** property to return the **Borders** collection, which contains all four borders. The following example adds a double border to cell A1 on worksheet one.

```
Worksheets(1).Range("A1").Borders.LineStyle = xlDouble
```

Use **Borders(index)**, where *index* identifies the border, to return a single **Border** object. The following example sets the color of the bottom border of cells A1:G1 to red.

```
Worksheets("Sheet1").Range("A1:G1")._
    Borders(xlEdgeBottom).Color = RGB(255, 0, 0)
```

*Index* can be one of the following **XlBordersIndex** constants: **xlDiagonalDown**, **xlDiagonalUp**, **xlEdgeBottom**, **xlEdgeLeft**, **xlEdgeRight**, or **xlEdgeTop**, **xlInsideHorizontal**, or **xlInsideVertical**.

## Remarks

You can set border properties for an individual border only with **Range** and **Style** objects. Other bordered objects, such as check boxes and chart areas, have a border that's treated as a single entity, regardless of how many sides it has. For these objects, you must return and set properties for the entire border as a unit. For more information, see the [Border](#) object.



[Show All](#)

# CalculatedFields Collection Object

[CalculatedFields](#)  [PivotField](#)  
 Multiple objects

A collection of [PivotField](#) objects that represents all the calculated fields in the specified PivotTable report. For example, a report that contains Revenue and Expense fields could have a calculated field named “Profit” defined as the amount in the Revenue field minus the amount in the Expense field.

## Remarks

For [OLAP](#) data sources, you cannot set this collection, and it always returns **Nothing**.

## Using the CalculatedFields Collection

Use the [CalculatedFields](#) method to return the **CalculatedFields** collection. The following example deletes the calculated fields from the PivotTable report named "Pivot1".

```
For Each fld in _  
    Worksheets(1).PivotTables("Pivot1").CalculatedFields  
    fld.Delete  
Next
```

Use **CalculatedFields(index)**, where *index* is specified field's name or index number, to return a single **PivotField** object from the **CalculatedFields** collection.



# CalculatedItems Collection Object

[CalculatedItems](#)   └ [PivotItem](#)  
└ Multiple objects

A collection of [PivotItem](#) objects that represent all the calculated items in the specified PivotTable report. For example, a PivotTable report that contains January, February, and March items could have a calculated item named “FirstQuarter” defined as the sum of the amounts in January, February, and March.

## Using the CalculatedItems Collection

Use the [CalculatedItems](#) method to return the **CalculatedItems** collection. The following example creates a list of the calculated items in the first PivotTable report on worksheet one, along with their formulas.

```
Set pt = Worksheets(1).PivotTables(1)
For Each ci In pt.PivotFields("Sales").CalculatedItems
    r = r + 1
    With Worksheets(2)
        .Cells(r, 1).Value = ci.Name
        .Cells(r, 2).Value = ci.Formula
    End With
Next
```

Use **CalculatedFields**(*index*), where *index* is the name or index number of the field, to return a single **PivotField** object from the **CalculatedFields** collection.



[Show All](#)

# CalculatedMembers Collection

[PivotTable](#) └ [CalculatedMembers](#)  
└ [CalculatedMember](#)

A collection of all the [CalculatedMember](#) objects on the specified PivotTable. Each **CalculatedMember** object represents a calculated member or calculated measure.

## Using the CalculatedMembers collection

Use the [CalculatedMembers](#) property of the [PivotTable](#) object to return a **CalculatedMembers** collection. The following example adds a set to a PivotTable, assuming a PivotTable exists on the active worksheet.

```
Sub UseCalculatedMember()  
  
    Dim pvtTable As PivotTable  
  
    Set pvtTable = ActiveSheet.PivotTables(1)  
  
    pvtTable.CalculatedMembers.Add Name:="[Beef]", _  
        Formula:="'{[Product].[All Products].Children}'", _  
        Type:=xlCalculatedSet  
  
End Sub
```

**Note** For the **Add** method in the previous example, the **Formula** argument must have a valid MDX syntax statement. The **Name** argument has to be acceptable to the [Online Analytical Processing \(OLAP\)](#) provider and the **Type** argument has to be defined.



# ChartGroups Collection

[ChartGroups](#)   └─ [ChartGroup](#)  
└─ Multiple objects

A collection of all the [ChartGroup](#) objects in the specified chart. Each **ChartGroup** object represents one or more series plotted in a chart with the same format. A chart contains one or more chart groups, each chart group contains one or more series, and each series contains one or more points. For example, a single chart might contain both a line chart group, containing all the series plotted with the line chart format, and a bar chart group, containing all the series plotted with the bar chart format.

## Using the ChartGroups Collection

Use the **ChartGroups** method to return the **ChartGroups** collection. The following example displays the number of chart groups on embedded chart one on worksheet one.

```
MsgBox Worksheets(1).ChartObjects(1).Chart.ChartGroups.Count
```

Use **ChartGroups(index)**, where *index* is the chart-group index number, to return a single **ChartGroup** object. The following example adds drop lines to chart group one on chart sheet one.

```
Charts(1).ChartGroups(1).HasDropLines = True
```

If the chart has been activated, you can use **ActiveChart**:

```
Charts(1).Activate  
ActiveChart.ChartGroups(1).HasDropLines = True
```

Because the index number for a particular chart group can change if the chart format used for that group is changed, it may be easier to use one of the named chart group shortcut methods to return a particular chart group. The **PieGroups** method returns the collection of pie chart groups in a chart, the **LineGroups** method returns the collection of line chart groups, and so on. Each of these methods can be used with an index number to return a single **ChartGroup** object, or without an index number to return a **ChartGroups** collection. The following chart group methods are available:

- [AreaGroups](#) method
- [BarGroups](#) method
- [ColumnGroups](#) method
- [DoughnutGroups](#) method
- [LineGroups](#) method
- [PieGroups](#) method



# ChartObjects Collection Object

[ChartObjects](#)  Multiple objects

A collection of all the [ChartObject](#) objects on the specified chart sheet, dialog sheet, or worksheet. Each **ChartObject** object represents an embedded chart. The **ChartObject** object acts as a container for a [Chart](#) object. Properties and methods for the **ChartObject** object control the appearance and size of the embedded chart on the sheet.

## Using the ChartObjects Collection

Use the **ChartObjects** method to return the **ChartObjects** collection. The following example deletes all the embedded charts on the worksheet named "Sheet1."

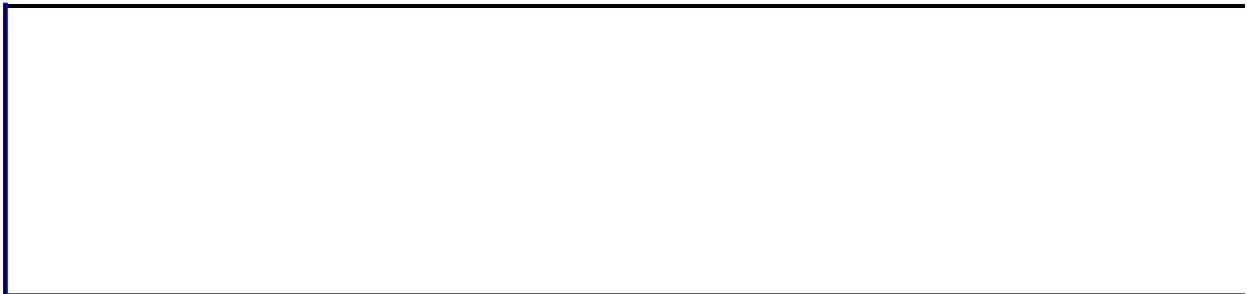
```
Worksheets("sheet1").ChartObjects.Delete
```

Use the [Add](#) method to create a new, empty embedded chart and add it to the collection. Use the [ChartWizard](#) method to add data and format the new chart. The following example creates a new embedded chart and then adds the data from cells A1:A20 as a line chart.

```
Dim ch As ChartObject  
Set ch = Worksheets("sheet1").ChartObjects.Add(100, 30, 400, 250)  
ch.Chart.ChartWizard source:=Worksheets("sheet1").Range("a1:a20"), _  
    gallery:=xlLine, title:="New Chart"
```

Use **ChartObjects(index)**, where *index* is the embedded chart index number or name, to return a single **ChartObject** object. The following example sets the pattern for the chart area in embedded chart one on the worksheet named "Sheet1."

```
Worksheets("Sheet1").ChartObjects(1).Chart. _  
    ChartArea.Interior.Pattern = xlLightDown
```



# Charts Collection

[Charts](#) └ Multiple objects

A collection of all the chart sheets in the specified or active workbook. Each chart sheet is represented by a **Chart** object. This doesn't include charts embedded on worksheets or dialog sheets. For information about embedded charts, see the [Chart](#) or [ChartObject](#) object.

## Using the Charts Collection

Use the **Charts** property to return the **Charts** collection. The following example prints all chart sheets in the active workbook.

```
Charts.PrintOut
```

Use the [Add](#) method to create a new chart sheet and add it to the workbook. The following example adds a new chart sheet to the active workbook and places the new chart sheet immediately after the worksheet named Sheet1.

```
Charts.Add After:=Worksheets("Sheet1")
```

You can combine the **Add** method with the **ChartWizard** method to add a new chart that contains data from a worksheet. The following example adds a new line chart based on data in cells A1:A20 on the worksheet named Sheet1.

```
With Charts.Add  
    .ChartWizard source:=Worksheets("Sheet1").Range("A1:A20"), _  
        Gallery:=xlLine, Title:="February Data"  
End With
```

Use **Charts(index)**, where *index* is the chart-sheet index number or name, to return a single **Chart** object. The following example changes the color of series one on chart sheet one to red.

```
Charts(1).SeriesCollection(1).Interior.Color = RGB(255, 0, 0)
```

The [Sheets](#) collection contains all the sheets in the workbook (both chart sheets and worksheets). Use **Sheets(index)**, where *index* is the sheet name or number, to return a single sheet.



# Comments Collection Object

[Worksheet](#) └ [Comments](#)  
└ [Comment](#)  
└ [Shape](#)

A collection of cell comments. Each comment is represented by a [Comment](#) object.

## Using the Comments Collection

Use the **Comments** property to return the **Comments** collection. The following example hides all the comments on worksheet one.

```
Set cmt = Worksheets(1).Comments
For Each c In cmt
    c.Visible = False
Next
```

Use the [AddComment](#) method to add a comment to a range. The following example adds a comment to cell E5 on worksheet one.

```
With Worksheets(1).Range("e5").AddComment
    .Visible = False
    .Text "reviewed on " & Date
End With
```

Use **Comments(index)**, where *index* is the comment number, to return a single comment from the **Comments** collection. The following example hides comment two on worksheet one.

```
Worksheets(1).Comments(2).Visible = False
```



[Show All](#)

# CubeFields Collection Object

[PivotTable](#) └ [CubeFields](#)  
└ [CubeField](#)  
└ Multiple objects

A collection of all **CubeField** objects in a PivotTable report that is based on an [OLAP cube](#). Each **CubeField** object represents a hierarchy or measure field from the cube.

## Using the CubeFields Collection

Use the [CubeFields](#) property to return the **CubeFields** collection. The following example creates a list of cube field names of the data fields in the first OLAP-based PivotTable report on Sheet1.

```
Set objNewSheet = Worksheets.Add  
intRow = 1  
For Each objCubeFld In _  
    Worksheets("Sheet1").PivotTables(1).CubeFields  
    If objCubeFld.Orientation = xlDataField Then  
        objNewSheet.Cells(intRow, 1).Value = objCubeFld.Name  
        intRow = intRow + 1  
    End If  
Next objCubeFld
```

Use **CubeFields(index)**, where *index* is the cube field's index number, to return a single **CubeField** object. The following example determines the name of the second cube field in the first PivotTable report on the active worksheet.

```
strAlphaName = _  
    ActiveSheet.PivotTables(1).CubeFields(2).Name
```



# CustomProperties Collection

Multiple objects [↳ CustomProperties](#)  
[↳ CustomProperty](#)

A collection of **CustomProperty** objects that represent additional information. The information can be used as metadata for XML.

## Using the CustomProperties collection

Use the **Properties** property of the **SmartTag** object, or the **CustomProperties** property of the **Worksheet** object, to return a **CustomProperties** collection.

Once a **CustomProperties** collection is returned, you can add metadata to worksheets and smart tags depending on which you choose to work with.

To add metadata to a worksheet, use the [CustomProperties](#) property with the **Add** method.

The following example demonstrates this feature. In this example, Microsoft Excel adds identifier information to the active worksheet and returns the name and value to the user.

```
Sub CheckCustomProperties()  
    Dim wksSheet1 As Worksheet  
    Set wksSheet1 = Application.ActiveSheet  
    ' Add metadata to worksheet.  
    wksSheet1.CustomProperties.Add _  
        Name:="Market", Value:="Nasdaq"  
    ' Display metadata.  
    With wksSheet1.CustomProperties.Item(1)  
        MsgBox .Name & vbTab & .Value  
    End With  
End Sub
```

To add metadata to a smart tag, use the [Properties](#) property with the **Add** method.

The following example demonstrates this feature. In this example, Microsoft Excel adds a smart tag titled "MSFT" to cell A1, then adds extra metadata called "Market" with the value of "Nasdaq" to the smart tag and then returns the value of the property to the user. This example assumes the host system is connected to the Internet when running this code sample and the checked recognizer called "Stock Ticker Symbol Recognizer" is enabled for Microsoft Excel.

```
Sub UseProperties()  
  
    Dim strLink As String  
    Dim strType As String  
  
    ' Define smart tag variables.  
    strLink = "urn:schemas-microsoft-com:smarttags#stocktickerSymbol"  
    strType = "stockview"  
  
    Range("A1").Formula = "MSFT"  
  
    ' Add a property for MSFT smart tag and define its value.  
    Range("A1").SmartTags.Add(strLink).Properties.Add _  
        Name:="Market", Value:="Nasdaq"  
  
    ' Notify the user of the smart tag's value.  
    MsgBox Range("A1").SmartTags.Add(strLink).Properties("Market").v  
  
End Sub
```



# CustomViews Collection Object

[Workbook](#) └ [CustomViews](#)  
└ [CustomView](#)

A collection of custom workbook views. Each view is represented by a [CustomView](#) object.

## Using the CustomViews Collection

Use the **CustomViews** property to return the **CustomViews** collection. Use the [Add](#) method to create a new custom view and add it to the **CustomViews** collection. The following example creates a new custom view named "Summary."

```
ActiveWorkbook.CustomViews.Add "Summary", True, True
```



# DataLabels Collection Object

[DataLabels](#)  Multiple objects

A collection of all the [DataLabel](#) objects for the specified series. Each **DataLabel** object represents a data label for a point or trendline. For a series without definable points (such as an area series), the **DataLabels** collection contains a single data label.

## Using the Datalabels Collection

Use the **DataLabels** method to return the **DataLabels** collection. The following example sets the number format for data labels on series one on chart sheet one.

```
With Charts(1).SeriesCollection(1)
    .HasDataLabels = True
    .DataLabels.NumberFormat = "##.##"
End With
```

Use **DataLabels(index)**, where *index* is the data-label index number, to return a single **DataLabel** object. The following example sets the number format for the fifth data label in series one in embedded chart one on worksheet one.

```
Worksheets(1).ChartObjects(1).Chart _
    .SeriesCollection(1).DataLabels(5).NumberFormat = "0.000"
```



# DiagramNodes Collection

[Diagram](#) └ [DiagramNodes](#)  
└ [DiagramNode](#)  
└ Multiple objects

A collection of [DiagramNode](#) objects that represents all the nodes in a diagram.

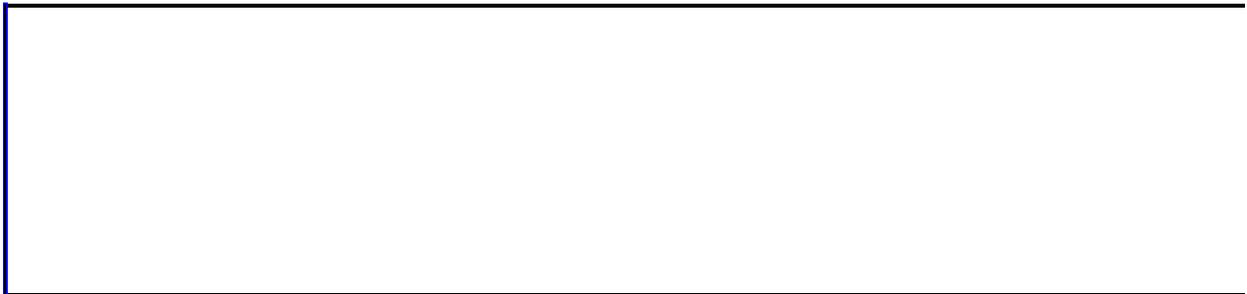
## Using the DiagramNodes collection

Use the [Nodes](#) property of the **Diagram** object to return a **DiagramNodes** collection. Use the [Item](#) method to select and work with a single diagram node in a diagram. This example assumes the first shape in the active worksheet is a diagram, selects the first node, and deletes it.

```
Sub FillDiagramNode()  
    ActiveSheet.Shapes(1).Diagram.Nodes.Item(1).Delete  
End Sub
```

Use the [SelectAll](#) method to select and work with all nodes in a diagram. This example assumes the first shape in the active worksheet is a diagram, selects all nodes, and fills them with the specified pattern.

```
Sub FillDiagramNodes()  
    ActiveSheet.Shapes(1).Diagram.Nodes.SelectAll  
    Selection.ShapeRange.Fill.Patterned msoPatternSmallConfetti  
End Sub
```



# Dialogs Collection Object

[Application](#) └ [Dialogs](#)  
└ [Dialog](#)

A collection of all the **Dialog** objects in Microsoft Excel. Each **Dialog** object represents a built-in dialog box. You cannot create a new built-in dialog box or add one to the collection. The only useful thing you can do with a **Dialog** object is use it with the [Show](#) method to display the dialog corresponding dialog box.

## Using the Dialogs Collection

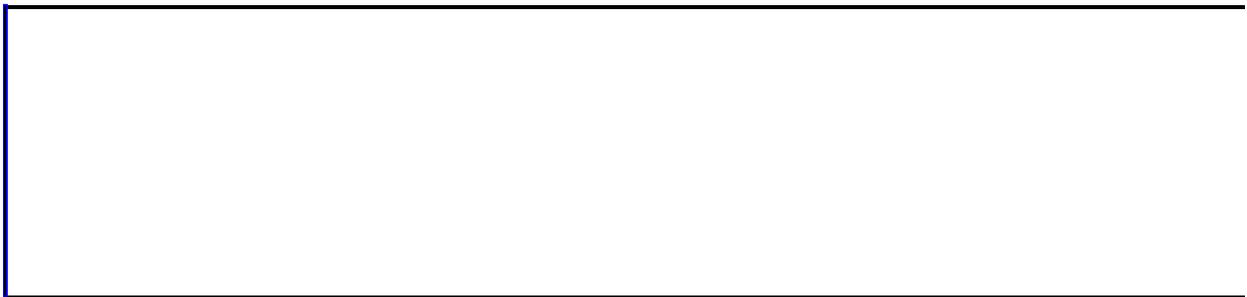
Use the [Dialogs](#) property to return the **Dialogs** collection. The following example displays the number of available built-in Microsoft Excel dialog boxes.

```
MsgBox Application.Dialogs.Count
```

Use **Dialogs(index)**, where *index* is a built-in constant identifying the dialog box, to return a single **Dialog** object. The following example runs the built-in **File Open** dialog box.

```
dlgAnswer = Application.Dialogs(xlDialogOpen).Show
```

The Microsoft Excel Visual Basic object library includes built-in constants for many of the built-in dialog boxes. Each constant is formed from the prefix "xlDialog" followed by the name of the dialog box. For example, the **Apply Names** dialog box constant is **xlDialogApplyNames**, and the **Find File** dialog box constant is **xlDialogFindFile**. These constants are members of the **XlBuiltinDialog** enumerated type. For more information about the available constants, see [Built-in Dialog Box Argument Lists](#).



# Errors Object

[Range](#) | [Errors](#)  
| [Error](#)

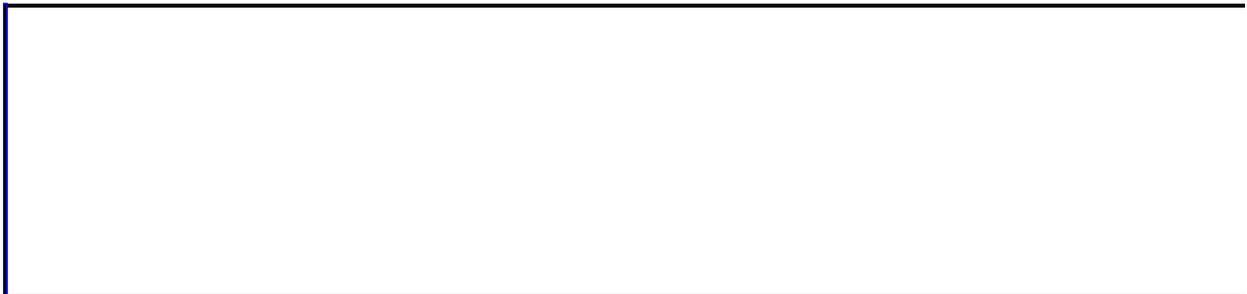
Represents the various spreadsheet errors for a range.

## Using the Errors object

Use the **Errors** property of the **Range** collection to return an **Errors** object.

Once an **Errors** object is returned, you can use the **Value** property of the [Error](#) object to check for particular error-checking conditions. The following example places a number as text in cell A1 and then notifies the user when the value of cell A1 contains a number as text.

```
Sub ErrorValue()  
    ' Place a number written as text in cell A1.  
    Range("A1").Formula = "'1"  
  
    If Range("A1").Errors.Item(xlNumberAsText).Value = True Then  
        MsgBox "Cell A1 has a number as text."  
    Else  
        MsgBox "Cell A1 is a number."  
    End If  
  
End Sub
```



# Filters Collection Object

[AutoFilter](#) └ [Filters](#)  
└ [Filter](#)

A collection of [Filter](#) objects that represents all the filters in an autofiltered range.

## Using the Filters Collection

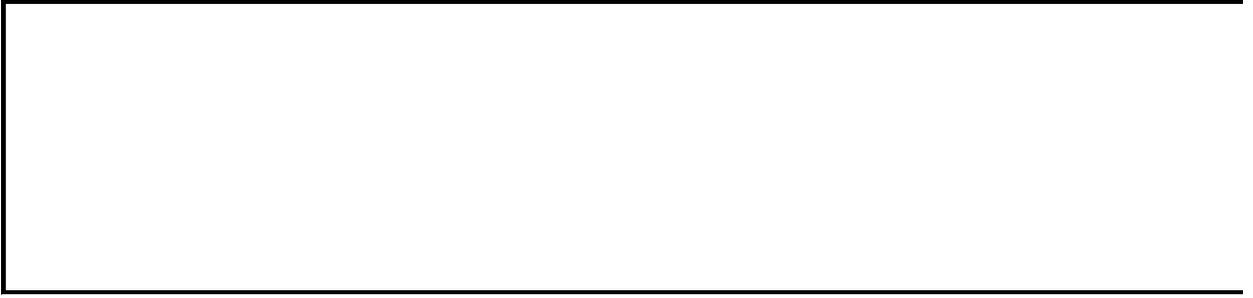
Use the **Filters** method to return the **Filters** collection. The following example creates a list that contains the criteria and operators for the filters in the autofiltered range on the Crew worksheet.

```
Dim f As Filter
Dim w As Worksheet
Const ns As String = "Not set"

Set w = Worksheets("Crew")
Set w2 = Worksheets("FilterData")
rw = 1
For Each f In w.AutoFilter.Filters
    If f.On Then
        c1 = Right(f.Criteria1, Len(f.Criteria1) - 1)
        If f.Operator Then
            op = f.Operator
            c2 = Right(f.Criteria2, Len(f.Criteria2) - 1)
        Else
            op = ns
            c2 = ns
        End If
    Else
        c1 = ns
        op = ns
        c2 = ns
    End If
    w2.Cells(rw, 1) = c1
    w2.Cells(rw, 2) = op
    w2.Cells(rw, 3) = c2
    rw = rw + 1
Next
```

Use **Filters(index)**, where *index* is the filter title or index number, to return a single **Filter** object. The following example sets a variable to the value of the **On** property of the filter for the first column in the filtered range on the Crew worksheet.

```
Set w = Worksheets("Crew")
If w.AutoFilterMode Then
    filterIsOn = w.AutoFilter.Filters(1).On
End If
```



# FormatConditions Collection Object

[Range](#) └ [FormatConditions](#)  
└ [FormatCondition](#)  
└ Multiple objects

Represents the collection of conditional formats for a single range. The **FormatConditions** collection can contain up to three conditional formats. Each format is represented by a [FormatCondition](#) object.

## Using the FormatConditions Collection

Use the **FormatConditions** property to return a **FormatConditions** object. Use the [Add](#) method to create a new conditional format, and use the [Modify](#) method to change an existing conditional format.

The following example adds a conditional format to cells E1:E10.

```
With Worksheets(1).Range("e1:e10").FormatConditions _
    .Add(xlCellValue, xlGreater, "=$a$1")
    With .Borders
        .LineStyle = xlContinuous
        .Weight = xlThin
        .ColorIndex = 6
    End With
    With .Font
        .Bold = True
        .ColorIndex = 3
    End With
End With
```

## Remarks

If you try to create more than three conditional formats for a single range, the **Add** method fails. If a range has three formats, you can use the **Modify** method to change one of the formats, or you can use the **Delete** method to delete a format and then use the **Add** method to create a new format.

For more information about conditional formats, see the [FormatCondition](#) object.



# HPageBreaks Collection Object

Multiple objects [└ HPageBreaks](#)

[└ HPageBreak](#)

[└ Multiple objects](#)

The collection of horizontal page breaks within the print area. Each horizontal page break is represented by an [HPageBreak](#) object.

## Using the HPageBreaks Collection

Use the [HPageBreaks](#) property to return the **HPageBreaks** collection. Use the [Add](#) method to add a horizontal page break. The following example adds a horizontal page break above the active cell.

```
ActiveSheet.HPageBreaks.Add Before:=ActiveCell
```

If you add a page break that does not intersect the print area, then the newly-added **HPageBreak** object will not appear in the **HPageBreaks** collection for the print area. The contents of the collection may change if the print area is resized or redefined.

When the [Application](#) property, [Count](#) property, [Creator](#) property, [Item](#) property, [Parent](#) property or [Add](#) method is used in conjunction with the **HPageBreaks** property:

- For an automatic print area, the **HPageBreaks** property applies only to the page breaks within the print area.
- For a user-defined print area of the same range, the **HPageBreaks** property applies to all of the page breaks.

**Note** There is a limit of 1026 horizontal page breaks per sheet.



# Hyperlinks Collection

Multiple objects [Hyperlinks](#)  
└─ [Hyperlink](#)  
    └─ Multiple objects

Represents the collection of hyperlinks for a worksheet or range. Each hyperlink is represented by a [Hyperlink](#) object.

## Using the Hyperlinks Collection

Use the [Hyperlinks](#) property to return the **Hyperlinks** collection. The following example checks the hyperlinks on worksheet one for a link that contains the word Microsoft.

```
For Each h in Worksheets(1).Hyperlinks
    If Instr(h.Name, "Microsoft") <> 0 Then h.Follow
Next
```

Use the [Add](#) method to create a hyperlink and add it to the **Hyperlinks** collection. The following example creates a new hyperlink for cell E5.

```
With Worksheets(1)
    .Hyperlinks.Add .Range("E5"), "http://example.microsoft.com"
End With
```



# LegendEntries Collection Object

[LegendEntries](#)    [LegendEntry](#)  
└─Multiple objects

A collection of all the [LegendEntry](#) objects in the specified chart legend. Each legend entry has two parts: the text of the entry, which is the name of the series or trendline associated with the legend entry; and the entry marker, which visually links the legend entry with its associated series or trendline in the chart. The formatting properties for the entry marker and its associated series or trendline are contained in the [LegendKey](#) object.

## Using the LegendEntries Collection

Use the **LegendEntries** method to return the **LegendEntries** collection. The following example loops through the collection of legend entries in embedded chart one and changes their font color.

```
With Worksheets("sheet1").ChartObjects(1).Chart.Legend
    For i = 1 To .LegendEntries.Count
        .LegendEntries(i).Font.ColorIndex = 5
    Next
End With
```

Use **LegendEntries(index)**, where *index* is the legend entry index number, to return a single **LegendEntry** object. You cannot return legend entries by name.

The index number represents the position of the legend entry in the legend. LegendEntries(1) is at the top of the legend; LegendEntries(LegendEntries.Count) is at the bottom. The following example changes the font style for the text of the legend entry at the top of the legend (this is usually the legend for series one) in embedded chart one to italic.

```
Worksheets("sheet1").ChartObjects(1).Chart _
    .Legend.LegendEntries(1).Font.Italic = True
```



[Show All](#)

# ListColumns Collection

[ListObject](#) └ [ListColumns](#)  
└ [ListColumn](#)  
└ Multiple objects

A collection of all the [ListColumn](#) objects in the specified **ListObject** object. Each **ListColumn** object represents a column in the [list](#).

## Using the ListColumns Collection

Use the [ListColumns](#) property of the [ListObject](#) object to return the **ListColumns** collection. The following example adds a new column to the default **ListObject** object in the first worksheet of the workbook. Because no position is specified, a new rightmost column is added.

```
Set myNewColumn = Worksheets(1).ListObject(1).ListColumns.Add
```

**Note** A name for the column is automatically generated. You can change the name after the column has been added.



# ListObjects Collection

[Worksheet](#) └ [ListObjects](#)  
└ [ListObject](#)  
└ Multiple objects

A collection of all the [ListObject](#) objects on a worksheet. Each **ListObject** object represents a list in the worksheet.

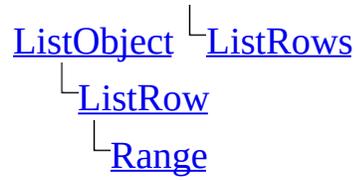
## Using the ListObjects Collection

Use the [ListObjects](#) property of the [Worksheet](#) object to return the **ListObjects** collection. The following example creates a new **ListObjects** collection which represents all the lists in a worksheet.

```
Set myWorksheetLists = Worksheets(1).ListObjects
```



# ListRows Collection



A collection of all the **ListRow** objects in the specified **ListObject** object. Each **ListRow** object represents a row in the list.

## Using the ListRows Collection

Use the [ListRows](#) property of the [ListObject](#) object to return the **ListRows Object** collection. The following example adds a new row to the default **ListObject** object in the first worksheet of the workbook. Because no position is specified, a new row is added to the end of the list.

```
Set myNewRow = Worksheets(1).ListObject(0).ListRows.Add
```



# Names Collection Object

Multiple objects [Names](#)  
└─ [Name](#)  
    └─ [Range](#)

A collection of all the [Name](#) objects in the application or workbook. Each **Name** object represents a defined name for a range of cells. Names can be either built-in names— such as Database, Print\_Area, and Auto\_Open— or custom names.

## Using the Names Collection

Use the **Names** property to return the **Names** collection. The following example creates a list of all the names in the active workbook, plus the addresses they refer to.

```
Set nms = ActiveWorkbook.Names
Set wks = Worksheets(1)
For r = 1 To nms.Count
    wks.Cells(r, 2).Value = nms(r).Name
    wks.Cells(r, 3).Value = nms(r).RefersToRange.Address
Next
```

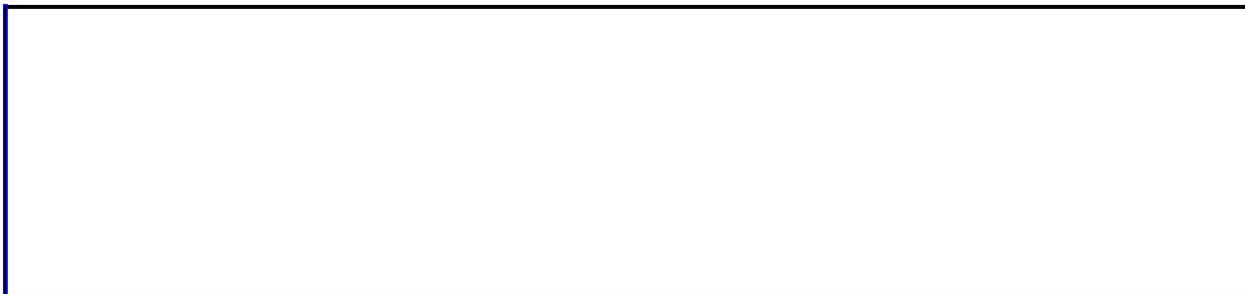
Use the [Add](#) method to create a name and add it to the collection. The following example creates a new name that refers to cells A1:C20 on the worksheet named "Sheet1."

```
Names.Add Name:="test", RefersTo:="=sheet1!$a$1:$c$20"
```

The **RefersTo** argument must be specified in A1-style notation, including dollar signs (\$) where appropriate. For example, if cell A10 is selected on Sheet1 and you define a name by using the **RefersTo** argument "=sheet1!A1:B1", the new name actually refers to cells A10:B10 (because you specified a relative reference). To specify an absolute reference, use "=sheet1!\$A\$1:\$B\$1".

Use **Names(index)**, where *index* is the name index number or defined name, to return a single **Name** object. The following example deletes the name "mySortRange" from the active workbook.

```
ActiveWorkbook.Names("mySortRange").Delete
```



# ODBCErrors Collection Object

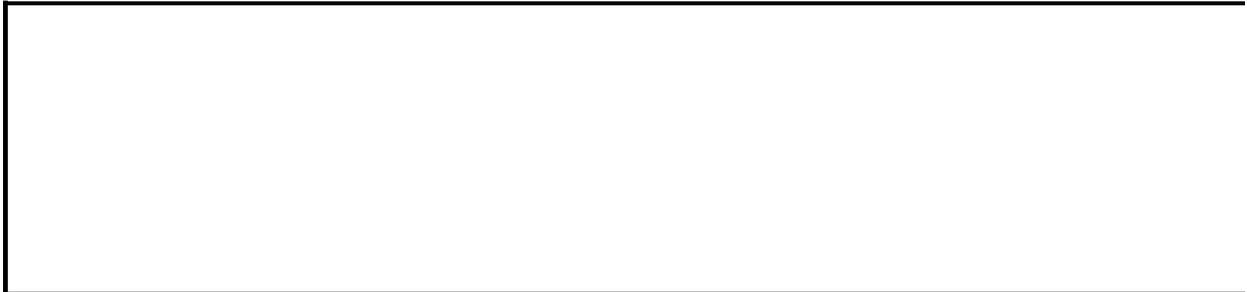
[Application](#) └ [ODBCErrors](#)  
└ [ODBCError](#)

A collection of [ODBCError](#) objects. Each **ODBCError** object represents an error returned by the most recent ODBC query. If the specified ODBC query runs without error, the **ODBCErrors** collection is empty. The errors in the collection are indexed in the order in which they're generated by the ODBC data source. You cannot add members to the collection.

## Using the ODBCErrors Collection

Use the **ODBCErrors** property to return the **ODBCErrors** collection. The following example refreshes query table one and displays any ODBC errors that occur.

```
With Worksheets(1).QueryTables(1)
    .Refresh
    Set errs = Application.ODBCErrors
    If errs.Count > 0 Then
        Set r = .Destination.Cells(1)
        r.Value = "The following errors occurred:"
        c = 0
        For Each er In errs
            c = c + 1
            r.Offset(c, 0).Value = er.ErrorString
            r.Offset(c, 1).Value = er.SqlState
        Next
    Else
        MsgBox "Query complete: all records returned."
    End If
End With
```



# OLEDBErrors Collection Object

[Application](#) └ [OLEDBErrors](#)  
└ [OLEDBError](#)

A collection of **OLEDBError** objects. Each **OLEDBError** object represents an error returned by the most recent OLE DB query. If the specified OLE DB query runs without error, the **OLEDBErrors** collection is empty. The errors in the collection are indexed in the order in which they're generated by the OLE DB provider. You cannot add members to the collection.

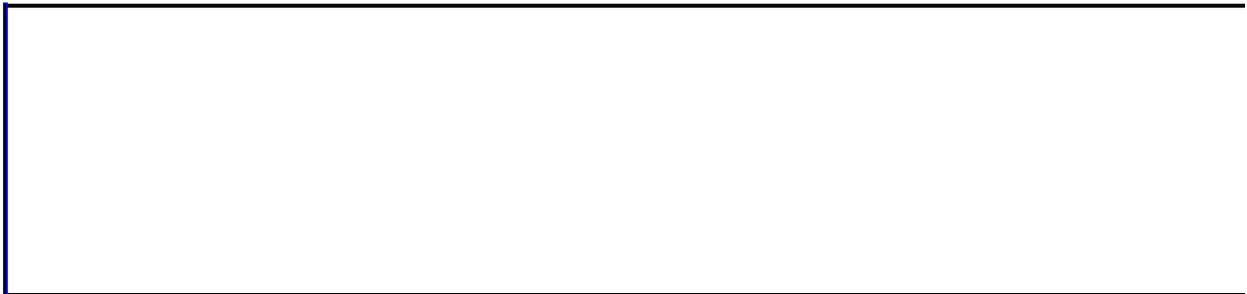
## Using the OLEDBErrors Collection

Use the [OLEDBErrors](#) property to return the **OLEDBErrors** collection. The following example displays the error description and the [SqlState](#) property's value for each OLE DB error in the collection.

```
For Each objEr in Application.OLEDBErrors
    MsgBox "The following error occurred:" & _
        objEr.ErrorString & " : " & objEr.SqlState
Next objEr
```

Use **OLEDBErrors(index)**, where *index* is the index number of the OLE DB error, to return a single **OLEDBError** object. The following example displays the error description and the [SqlState](#) property's value for the first error returned by the most recent OLE DB query.

```
Set objEr = Application.OLEDBErrors(1)
MsgBox "The following error occurred:" & _
    objEr.ErrorString & " : " & objEr.SqlState
```



# OLEObjects Collection Object

[OLEObjects](#) └ Multiple objects

A collection of all the [OLEObject](#) objects on the specified worksheet. Each **OLEObject** object represents an ActiveX control or a linked or embedded OLE object.

## Using the OLEObjects Collection

Use the [OLEObjects](#) method to return the **OLEObjects** collection. The following example hides all the OLE objects on worksheet one.

```
Worksheets(1).OLEObjects.Visible = False
```

Use the [Add](#) method to create a new OLE object and add it to the **OLEObjects** collection. The following example creates a new OLE object representing the bitmap file Arcade.bmp and adds it to worksheet one.

```
Worksheets(1).OLEObjects.Add FileName:="arcade.gif"
```

The following example creates a new ActiveX control (a list box) and adds it to worksheet one.

```
Worksheets(1).OLEObjects.Add ClassType:="Forms.ListBox.1"
```

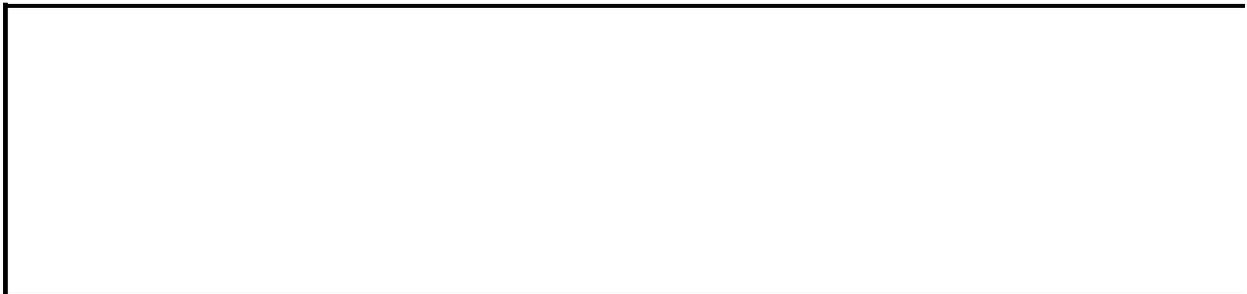
For more information, see [Using ActiveX controls on sheets](#).

## Remarks

An ActiveX control on a sheet has two names: the name of the shape that contains the control, which you can see in the **Name** box when you view the sheet, and the code name for the control, which you can see in the cell to the right of **(Name)** in the **Properties** window. When you first add a control to a sheet, the shape name and code name match. However, if you change either the shape name or code name, the other is not automatically changed to match.

You use the code name of a control in the names of its event procedures. However, when you return a control from the **Shapes** or **OLEObjects** collection for a sheet, you must use the shape name, not the code name, to refer to the control by name. For example, assume that you add a check box to a sheet and that both the default shape name and the default code name are `CheckBox1`. If you then change the control code name by typing `chkFinished` next to **(Name)** in the **Properties** window, you must use `chkFinished` in event procedures names, but you still have to use `CheckBox1` to return the control from the **Shapes** or **OLEObject** collection, as shown in the following example.

```
Private Sub chkFinished_Click()  
    ActiveSheet.OLEObjects("CheckBox1").Object.Value = 1  
End Sub
```



# Panes Collection Object

[Window](#) └ [Panes](#)  
└ [Pane](#)  
└ [Range](#)

A collection of all the [Pane](#) objects shown in the specified window. **Pane** objects exist only for worksheets and Microsoft Excel 4.0 macro sheets.

## Using the Panes Collection

Use the **Panes** property to return the **Panes** collection. The following example freezes panes in the active window if the window contains more than one pane.

```
If ActiveWindow.Panes.Count > 1 Then _  
    ActiveWindow.FreezePanes = True
```

Use **Panes(index)**, where *index* is the pane index number, to return a single **Pane** object. The following example scrolls through the upper-left pane of the window in which Sheet1 is displayed.

```
Worksheets("sheet1").Activate  
Windows(1).Panes(1).LargeScroll down:=1
```



# Parameters Collection Object

[QueryTable](#) └ [Parameters](#)  
└ [Parameter](#)  
└ [Range](#)

A collection of [Parameter](#) objects for the specified query table. Each **Parameter** object represents a single query parameter. Every query table contains a **Parameters** collection, but the collection is empty unless the query table is using a parameter query.

## Using the Parameters Collection

Use the **Parameters** property to return the **Parameters** collection. The following example displays the number of parameters in query table one.

```
MsgBox Worksheets(1).ActiveSheet.QueryTables(1).Parameters.Count
```

Use the [Add](#) method to create a new parameter for a query table. The following example changes the SQL statement for query table one. The clause “(city=?)” indicates that the query is a parameter query, and the value of city is set to the constant “Oakland.”

```
Set qt = Sheets("sheet1").QueryTables(1)
qt.Sql = "SELECT * FROM authors WHERE (city=?)"
Set param1 = qt.Parameters.Add("City Parameter", _
    xlParamTypeVarChar)
param1.SetParam xlConstant, "Oakland"
qt.Refresh
```

You cannot use the **Add** method on a URL connection query table. For URL connection query tables, Microsoft Excel creates the parameters based on the [Connection](#) and [PostText](#) properties.



# Phonetics Collection Object

[Range](#) └ [Phonetics](#)  
└ [Font](#)

A collection of all the [Phonetic](#) objects in the specified range. Each **Phonetic** object contains information about a specific phonetic text string.

## Using the Phonetics Collection

Use the [Phonetics](#) property to return the **Phonetics** collection. The following example makes all phonetic text in the range A1:C4 visible.

```
Range("A1:C4").Phonetics.Visible = True
```

Use **Phonetics**(*index*), where *index* is the index number of the phonetic text, to return a single **Phonetic** object. The following example sets the first phonetic text string in the active cell to "フリカダ".

```
ActiveCell.Phonetics(1).Text = "フリカダ"
```



# PivotCaches Collection Object

[PivotCaches](#)  [PivotCache](#)

Represents the collection of memory caches from the PivotTable reports in a workbook. Each memory cache is represented by a [PivotCache](#) object.

## Using the PivotCaches Collection

Use the [PivotCaches](#) method to return the **PivotCaches** collection. The following example sets the **RefreshOnFileOpen** property for all memory caches in the active workbook.

```
For Each pc In ActiveWorkbook.PivotCaches  
    pc.RefreshOnFileOpen = True  
Next
```



# PivotFields Collection Object

[CubeField](#) └ [PivotFields](#)  
└ [PivotTable](#)

A collection of all the [PivotField](#) objects in a PivotTable report.

## Using the PivotFields Collection

Use the **PivotFields** method of the **PivotTable** object to return the **PivotFields** collection. The following example enumerates the field names in the first PivotTable report on Sheet3.

```
With Worksheets("sheet3").PivotTables(1)
    For i = 1 To .PivotFields.Count
        MsgBox .PivotFields(i).Name
    Next
End With
```

Use **PivotFields(index)**, where *index* is the field name or index number, to return a single **PivotField** object. The following example makes the Year field a row field in the first PivotTable report on Sheet3.

```
Worksheets("sheet3").PivotTables(1) _
    .PivotFields("year").Orientation = xlRowField
```

In some cases, it may be easier to use one of the properties that returns a subset of the PivotTable fields. The following accessor methods are available:

- [ColumnFields](#) property
- [DataFields](#) property
- [HiddenFields](#) property
- [PageFields](#) property
- [RowFields](#) property
- [VisibleFields](#) property



[Show All](#)

# PivotFormulas Collection Object

[PivotTable](#) └ [PivotFormulas](#)  
└ [PivotFormula](#)

Represents the collection of formulas for a PivotTable report. Each formula is represented by a **PivotFormula** object.

## Remarks

This object and its associated properties and methods aren't available for [OLAP](#) data sources because calculated fields and items aren't supported.

## Using the PivotFormulas Collection

Use the [PivotFormulas](#) method to return the **PivotFormulas** collection. The following example creates a list of PivotTable formulas for the first PivotTable report on the active worksheet.

```
For Each pf in ActiveSheet.PivotTables(1).PivotFormulas
    Cells(r, 1).Value = pf.Formula
    r = r + 1
Next
```



# PivotItemList Collection

[PivotCell](#) └ [PivotItemList](#)  
└ [PivotItem](#)  
└ Multiple objects

A collection of all the **PivotItem** objects in the specified PivotTable. Each **PivotItem** represents an item in a PivotTable field.

## Using the PivotItemList collection

Use the [RowItems](#) or [ColumnItems](#) property of the [PivotCell](#) object to return a **PivotItemList** collection.

Once a **PivotItemList** collection is returned, you can use the [Item](#) method to identify a particular **PivotItem** list. The following example displays the **PivotItem** list associated with cell B5 to the user. This example assumes a PivotTable exists on the active worksheet.

```
Sub CheckPivotItemList()  
    ' Identify contents associated with PivotItemList.  
    MsgBox "Contents associated with cell B5: " & _  
        Application.Range("B5").PivotCell.RowItems.Item(1)  
End Sub
```



# PivotItems Collection Object

[PivotItems](#)  [PivotField](#)

A collection of all the [PivotItem](#) objects in a PivotTable field. The items are the individual data entries in a field category.

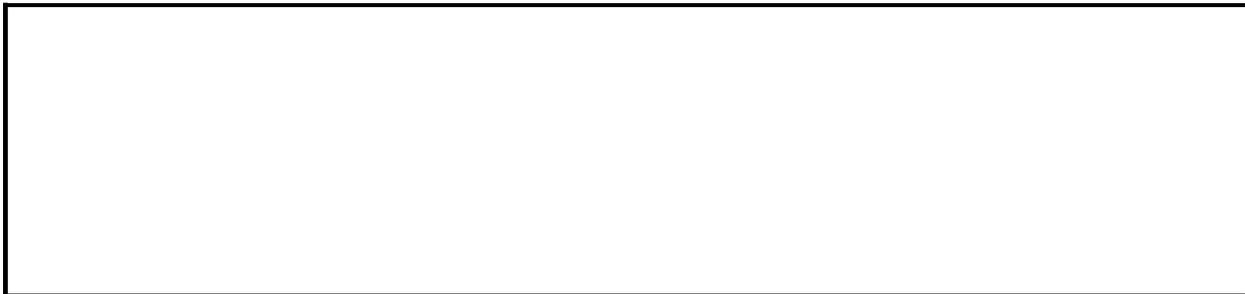
## Using the PivotItems Collection

Use the **PivotItems** method to return the **PivotItems** collection. The following example creates an enumerated list of field names and the items contained in those fields for the first PivotTable report on Sheet4.

```
Worksheets("sheet4").Activate
With Worksheets("sheet3").PivotTables(1)
    c = 1
    For i = 1 To .PivotFields.Count
        r = 1
        Cells(r, c) = .PivotFields(i).Name
        r = r + 1
        For x = 1 To .PivotFields(i).PivotItems.Count
            Cells(r, c) = .PivotFields(i).PivotItems(x).Name
            r = r + 1
        Next
        c = c + 1
    Next
End With
```

Use **PivotItems(index)**, where *index* is the item index number or name, to return a single **PivotItem** object. The following example hides all entries in the first PivotTable report on Sheet3 that contain "1998" in the Year field.

```
Worksheets("sheet3").PivotTables(1) _
    .PivotFields("year").PivotItems("1998").Visible = False
```



# PivotTables Collection Object

[PivotTables](#)   └ [PivotTable](#)  
└ Multiple objects

A collection of all the [PivotTable](#) objects on the specified worksheet.

## Using the PivotTables Collection

Use the **PivotTables** method to return the **PivotTables** collection. The following example displays the number of PivotTable reports on Sheet3.

```
MsgBox Worksheets("sheet3").PivotTables.Count
```

Use the [PivotTableWizard](#) method to create a new PivotTable report and add it to the collection. The following example creates a new PivotTable report from a Microsoft Excel database (contained in the range A1:C100).

```
ActiveSheet.PivotTableWizard xlDatabase, Range("A1:C100")
```

Use **PivotTables(index)**, where *index* is the PivotTable index number or name, to return a single **PivotTable** object. The following example makes the Year field a row field in the first PivotTable report on Sheet3.

```
Worksheets("sheet3").PivotTables(1) _  
    .PivotFields("year").Orientation = xlRowField
```

## Remarks

Because PivotTable report programming can be complex, it's generally easiest to record PivotTable report actions and then revise the recorded code. To record a macro, point to **Macro** on the **Tools** menu and click **Record New Macro**.

---

# Points Collection Object

[Points](#)  [Point](#)  
 Multiple objects

A collection of all the [Point](#) objects in the specified series in a chart.

## Using the Points Collection

Use the **Points** method to return the **Points** collection. The following example adds a data label to the last point on series one in embedded chart one on worksheet one.

```
Dim pts As Points
Set pts = Worksheets(1).ChartObjects(1).Chart. _
    SeriesCollection(1).Points
pts(pts.Count).ApplyDataLabels type:=xlShowValue
```

Use **Points(index)**, where *index* is the point index number, to return a single **Point** object. Points are numbered from left to right on the series. **Points(1)** is the leftmost point, and **Points(Points.Count)** is the rightmost point. The following example sets the marker style for the third point in series one in embedded chart one on worksheet one. The specified series must be a 2-D line, scatter, or radar series.

```
Worksheets(1).ChartObjects(1).Chart. _
    SeriesCollection(1).Points(3).MarkerStyle = xlDiamond
```



# PublishObjects Collection Object

[Workbook](#) └ [PublishObjects](#)  
└ [PublishObject](#)

A collection of all [PublishObject](#) objects in the workbook. Each **PublishObject** object represents an item in a workbook that has been saved to a Web page and can be refreshed according to values specified by the properties and methods of the object.

## Using the PublishObjects Collection

Use the [PublishObjects](#) property to return the **PublishObjects** collection. The following example saves all static **PublishObject** objects in the active workbook to the Web page.

```
Set objPObj = ActiveWorkbook.PublishObjects
For Each objPO in objPObj
    If objPO.HtmlType = xlHTMLStatic Then
        objPO.Publish
    End If
Next objPO
```

Use **PublishObjects(index)**, where *index* is the index number of the specified item in the workbook, to return a single **PublishObject** object. The following example sets the location where the first item in workbook three is saved.

```
Workbooks(3).PublishObjects(1).FileName = _
    "\\myserver\public\finacct\statemnt.htm"
```



# QueryTables Collection Object

[Worksheet](#) └ [QueryTables](#)  
└ [QueryTable](#)  
└ Multiple objects

A collection of [QueryTable](#) objects. Each **QueryTable** object represents a worksheet table built from data returned from an external data source.

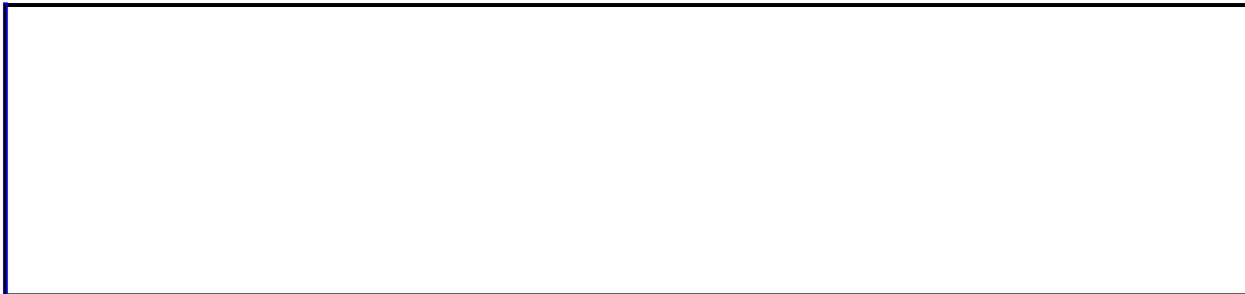
## Using the QueryTables Collection

Use the **QueryTables** property to return the **QueryTables** collection. The following example displays the number of query tables on the active worksheet.

```
MsgBox ActiveSheet.QueryTables.Count
```

Use the [Add](#) method to create a new query table and add it to the **QueryTables** collection. The following example creates a new query table.

```
Dim qt As QueryTable
sqlstring = "select 96Sales.totals from 96Sales where profit < 5"
connstring = _
    "ODBC;DSN=96SalesData;UID=Rep21;PWD=NUyHwYQI;Database=96Sales"
With ActiveSheet.QueryTables.Add(Connection:=connstring, _
    Destination:=Range("B1"), Sql:=sqlstring)
    .Refresh
End With
```



# Range Collection

Multiple objects  [Range](#)  
 Multiple objects

Represents a cell, a row, a column, a selection of cells containing one or more contiguous blocks of cells, or a 3-D range.

## Using the Range Collection

The following properties and methods for returning a **Range** object are described in this section:

- **Range** property
- **Cells** property
- **Range** and **Cells**
- **Offset** property
- **Union** method

## Range Property

Use **Range**(*arg*), where *arg* names the range, to return a **Range** object that represents a single cell or a range of cells. The following example places the value of cell A1 in cell A5.

```
Worksheets("Sheet1").Range("A5").Value = _  
    Worksheets("Sheet1").Range("A1").Value
```

The following example fills the range A1:H8 with random numbers by setting the formula for each cell in the range. When it's used without an object qualifier (an object to the left of the period), the **Range** property returns a range on the active sheet. If the active sheet isn't a worksheet, the method fails. Use the [Activate](#) method to activate a worksheet before you use the **Range** property without an explicit object qualifier.

```
Worksheets("Sheet1").Activate  
Range("A1:H8").Formula = "=Rand()"      'Range is on the active sheet
```

The following example clears the contents of the range named Criteria.

```
Worksheets(1).Range("Criteria").ClearContents
```

If you use a text argument for the range address, you must specify the address in A1-style notation (you cannot use R1C1-style notation).

## Cells Property

Use **Cells**(*row*, *column*) where *row* is the row index and *column* is the column index, to return a single cell. The following example sets the value of cell A1 to 24.

```
Worksheets(1).Cells(1, 1).Value = 24
```

The following example sets the formula for cell A2.

```
ActiveSheet.Cells(2, 1).Formula = "=Sum(B1:B5)"
```

Although you can also use `Range("A1")` to return cell A1, there may be times when the **Cells** property is more convenient because you can use a variable for the row or column. The following example creates column and row headings on Sheet1. Notice that after the worksheet has been activated, the **Cells** property can be used without an explicit sheet declaration (it returns a cell on the active sheet).

```
Sub SetUpTable()  
Worksheets("Sheet1").Activate  
For TheYear = 1 To 5  
    Cells(1, TheYear + 1).Value = 1990 + TheYear  
Next TheYear  
For TheQuarter = 1 To 4  
    Cells(TheQuarter + 1, 1).Value = "Q" & TheQuarter  
Next TheQuarter  
End Sub
```

Although you could use Visual Basic string functions to alter A1-style references, it's much easier (and much better programming practice) to use the `Cells(1, 1)` notation.

Use *expression*.**Cells**(*row*, *column*) , where *expression* is an expression that returns a **Range** object, and *row* and *column* are relative to the upper-left corner of the range, to return part of a range. The following example sets the formula for cell C5.

```
Worksheets(1).Range("C5:C10").Cells(1, 1).Formula = "=Rand()"
```

## Range and Cells

Use **Range**(*cell1*, *cell2*), where *cell1* and *cell2* are **Range** objects that specify the start and end cells, to return a **Range** object. The following example sets the border line style for cells A1:J10.

```
With Worksheets(1)
    .Range(.Cells(1, 1), _
        .Cells(10, 10)).Borders.LineStyle = xlThick
End With
```

Notice the period in front of each occurrence of the **Cells** property. The period is required if the result of the preceding **With** statement is to be applied to the **Cells** property— in this case, to indicate that the cells are on worksheet one (without the period, the **Cells** property would return cells on the active sheet).

## Offset Property

Use **Offset**(*row*, *column*), where *row* and *column* are the row and column offsets, to return a range at a specified offset to another range. The following example selects the cell three rows down from and one column to the right of the cell in the upper-left corner of the current selection. You cannot select a cell that isn't on the active sheet, so you must first activate the worksheet.

```
Worksheets("Sheet1").Activate  
    'Can't select unless the sheet is active  
Selection.Offset(3, 1).Range("A1").Select
```

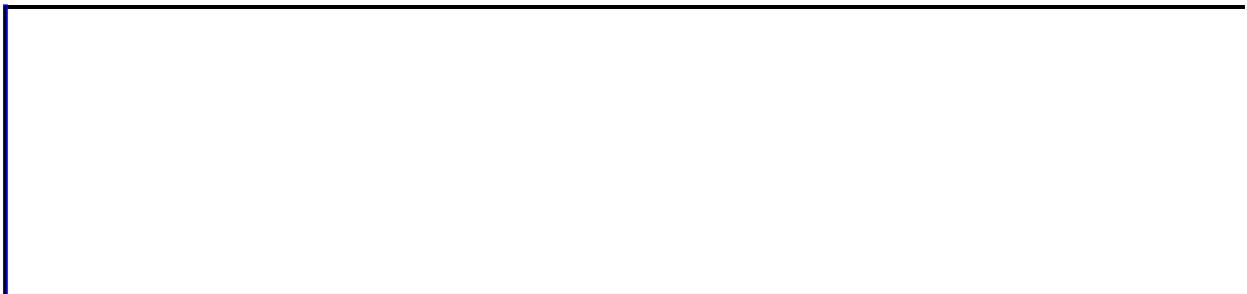
## Union Method

Use **Union**(*range1*, *range2*, ...) to return multiple-area ranges— that is, ranges composed of two or more contiguous blocks of cells. The following example creates an object defined as the union of ranges A1:B2 and C3:D4, and then selects the defined range.

```
Dim r1 As Range, r2 As Range, myMultiAreaRange As Range
Worksheets("sheet1").Activate
Set r1 = Range("A1:B2")
Set r2 = Range("C3:D4")
Set myMultiAreaRange = Union(r1, r2)
myMultiAreaRange.Select
```

If you work with selections that contain more than one area, the [Areas](#) property is very useful. It divides a multiple-area selection into individual **Range** objects and then returns the objects as a collection. You can use the [Count](#) property on the returned collection to check for a selection that contains more than one area, as shown in the following example.

```
Sub NoMultiAreaSelection()
    NumberOfSelectedAreas = Selection.Areas.Count
    If NumberOfSelectedAreas > 1 Then
        MsgBox "You cannot carry out this command " & _
            "on multi-area selections"
    End If
End Sub
```



# RecentFiles Collection Object

[Application](#) └ [RecentFiles](#)  
└ [RecentFile](#)

Represents the list of recently used files. Each file is represented by a [RecentFile](#) object.

## Using the RecentFiles Collection

Use the [RecentFiles](#) property to return the **RecentFiles** collection. The following example sets the maximum number of files in the list of recently used files.

```
Application.RecentFiles.Maximum = 6
```



# Scenarios Collection Object

[Scenarios](#)   └ [Scenario](#)  
          └ [Range](#)

A collection of all the [Scenario](#) objects on the specified worksheet. A scenario is a group of input values (called *changing cells*) that's named and saved.

## Using the Scenarios Collection

Use the **Scenarios** method to return the **Scenarios** collection. The following example creates a summary for the scenarios on the worksheet named "Options," using cells J10 and J20 as the result cells.

```
Worksheets("options").Scenarios.CreateSummary _  
    resultCells:=Worksheets("options").Range("j10,j20")
```

Use the [Add](#) method to create a new scenario and add it to the collection. The following example adds a new scenario named "Typical" to the worksheet named "Options." The new scenario has two changing cells, A2 and A12, with the respective values 55 and 60.

```
Worksheets("options").Scenarios.Add name:="Typical", _  
    changingCells:=Worksheets("options").Range("A2,A12"), _  
    values:=Array("55", "60")
```

Use **Scenarios(index)**, where *index* is the scenario name or index number, to return a single **Scenario** object. The following example shows the scenario named "Typical" on the worksheet named "Options."

```
Worksheets("options").Scenarios("typical").Show
```



# SeriesCollection Collection Object

[SeriesCollection](#)  [Series](#)  
 Multiple objects

A collection of all the [Series](#) objects in the specified chart or chart group.

## Using the SeriesCollection Collection

Use the **SeriesCollection** method to return the **SeriesCollection** collection. The following example adds the data in cells C1:C10 on worksheet one to an existing series in the series collection in embedded chart one.

```
Worksheets(1).ChartObjects(1).Chart. _  
    SeriesCollection.Extend Worksheets(1).Range("c1:c10")
```

Use the [Add](#) method to create a new series and add it to the chart. The following example adds the data from cells A1:A19 as a new series on the chart sheet named "Chart1."

```
Charts("chart1").SeriesCollection.Add _  
    source:=Worksheets("sheet1").Range("a1:a19")
```

Use **SeriesCollection(index)**, where *index* is the series index number or name, to return a single **Series** object. The following example sets the color of the interior for the first series in embedded chart one on Sheet1.

```
Worksheets("sheet1").ChartObjects(1).Chart. _  
    SeriesCollection(1).Interior.Color = RGB(255, 0, 0)
```



# ShapeNodes Collection Object

Multiple objects [↳ ShapeNodes](#)  
[↳ ShapeNode](#)

A collection of all the [ShapeNode](#) objects in the specified freeform. Each **ShapeNode** object represents either a node between segments in a freeform or a control point for a curved segment of a freeform. You can create a freeform manually or by using the [BuildFreeform](#) and [ConvertToShape](#) methods.

## Using the ShapeNodes Collection

Use the **Nodes** property to return the **ShapeNodes** collection. The following example deletes node four in shape three on myDocument. For this example to work, shape three must be a freeform with at least four nodes.

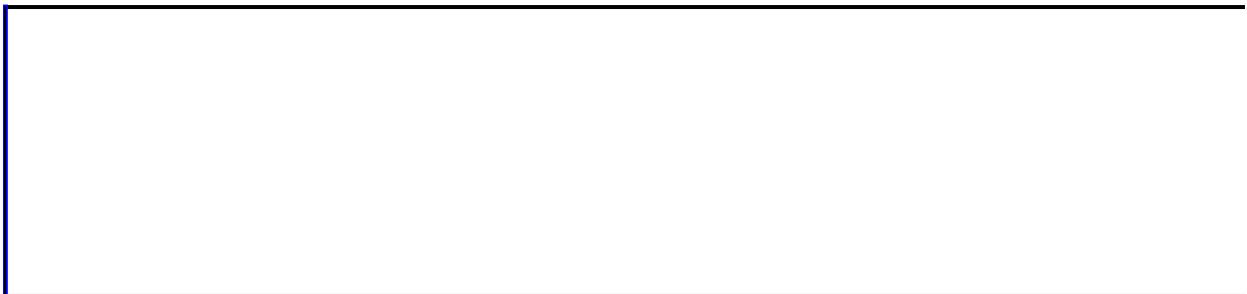
```
Set myDocument = Worksheets(1)
myDocument.Shapes(3).Nodes.Delete 4
```

Use the [Insert](#) method to create a new node and add it to the **ShapeNodes** collection. The following example adds a smooth node with a curved segment after node four in shape three on myDocument. For this example to work, shape three must be a freeform with at least four nodes.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(3).Nodes
    .Insert 4, msoSegmentCurve, msoEditingSmooth, 210, 100
End With
```

Use **Nodes(index)**, where *index* is the node index number, to return a single **ShapeNode** object. If node one in shape three on myDocument is a corner point, the following example makes it a smooth point. For this example to work, shape three must be a freeform.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(3)
    If .Nodes(1).EditingType = msoEditingCorner Then
        .Nodes.SetEditingType 1, msoEditingSmooth
    End If
End With
```



# ShapeRange Collection

Multiple objects [↳ ShapeRange](#)  
↳ Multiple objects

Represents a shape range, which is a set of shapes on a document. A shape range can contain as few as a single shape or as many as all the shapes on the document. You can include whichever shapes you want— chosen from among all the shapes on the document or all the shapes in the selection— to construct a shape range. For example, you could construct a **ShapeRange** collection that contains the first three shapes on a document, all the selected shapes on a document, or all the freeforms on a document.

For an overview of how to work with either a single shape or with more than one shape at a time, see [Working with Shapes \(Drawing Objects\)](#).

## Using the ShapeRange Collection

This section describes how to:

- Return a set of shapes you specify by name or index number.
- Return all or some of the selected shapes on a document.

## Returning a Set of Shapes You Specify by Name or Index Number

Use **Shapes.Range**(*index*), where *index* is the name or index number of the shape or an array that contains either names or index numbers of shapes, to return a **ShapeRange** collection that represents a set of shapes on a document. You can use the **Array** function to construct an array of names or index numbers. The following example sets the fill pattern for shapes one and three on myDocument.

```
Set myDocument = Worksheets(1)
myDocument.Shapes.Range(Array(1, 3)).Fill.Patterned _
    msoPatternHorizontalBrick
```

The following example sets the fill pattern for the shapes named Oval 4 and Rectangle 5 on myDocument.

```
Set myDocument = Worksheets(1)
Set myRange = myDocument.Shapes.Range(Array("Oval 4", _
    "Rectangle 5"))
myRange.Fill.Patterned msoPatternHorizontalBrick
```

Although you can use the [Range](#) property to return any number of shapes or slides, it's simpler to use the [Item](#) method if you want to return only a single member of the collection. For example, `Shapes(1)` is simpler than `Shapes.Range(1)`.

## Returning All or Some of the Selected Shapes on a Document

Use the [ShapeRange](#) property of the **Selection** object to return all the shapes in the selection. The following example sets the fill foreground color for all the shapes in the selection in window one, assuming that there's at least one shape in the selection.

```
Windows(1).Selection.ShapeRange.Fill.ForeColor.RGB = _  
    RGB(255, 0, 255)
```

Use **Selection.ShapeRange(index)**, where *index* is the shape name or the index number, to return a single shape within the selection. The following example sets the fill foreground color for shape two in the collection of selected shapes in window one, assuming that there are at least two shapes in the selection.

```
Windows(1).Selection.ShapeRange(2).Fill.ForeColor.RGB = _  
    RGB(255, 0, 255)
```



# Shapes Collection

Multiple objects [└ Shapes](#)  
[└ ShapeRange](#)

A collection of all the [Shape](#) objects on the specified sheet. Each **Shape** object represents an object in the drawing layer, such as an AutoShape, freeform, OLE object, or picture.

**Note** If you want to work with a subset of the shapes on a document— for example, to do something to only the AutoShapes on the document or to only the selected shapes— you must construct a [ShapeRange](#) collection that contains the shapes you want to work with. For an overview of how to work either with a single shape or with more than one shape at a time, see [Working with Shapes \(Drawing Objects\)](#).

## Using the Shapes Collection

Use the **Shapes** property to return the **Shapes** collection. The following example selects all the shapes on myDocument.

```
Set myDocument = Worksheets(1)
myDocument.Shapes.SelectAll
```

**Note** If you want to do something (like delete or set a property) to all the shapes on a sheet at the same time, select all the shapes and then use the **ShapeRange** property on the selection to create a **ShapeRange** object that contains all the shapes on the sheet, and then apply the appropriate property or method to the **ShapeRange** object.

Use **Shapes(index)**, where *index* is the shape's name or index number, to return a single **Shape** object. The following example sets the fill to a preset shade for shape one on myDocument.

```
Set myDocument = Worksheets(1)
myDocument.Shapes(1).Fill.PresetGradient _
    msoGradientHorizontal, 1, msoGradientBrass
```

Use **Shapes.Range(index)**, where *index* is the shape's name or index number or an array of shape names or index numbers, to return a **ShapeRange** collection that represents a subset of the **Shapes** collection. The following example sets the fill pattern for shapes one and three on myDocument.

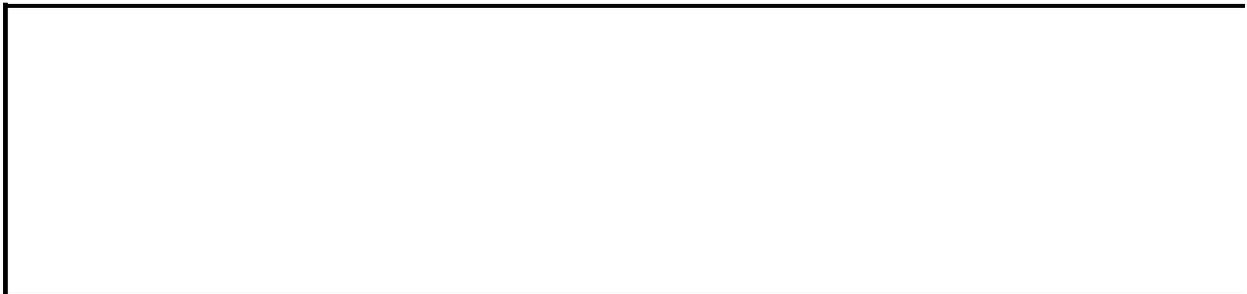
```
Set myDocument = Worksheets(1)
myDocument.Shapes.Range(Array(1, 3)).Fill.Patterned _
    msoPatternHorizontalBrick
```

## Remarks

An ActiveX control on a sheet has two names: the name of the shape that contains the control, which you can see in the **Name** box when you view the sheet, and the code name for the control, which you can see in the cell to the right of **(Name)** in the **Properties** window. When you first add a control to a sheet, the shape name and code name match. However, if you change either the shape name or code name, the other isn't automatically changed to match.

You use the code name of a control in the names of its event procedures. However, when you return a control from the **Shapes** or **OLEObjects** collection for a sheet, you must use the shape name, not the code name, to refer to the control by name. For example, assume that you add a check box to a sheet and that both the default shape name and the default code name are `CheckBox1`. If you then change the control code name by typing `chkFinished` next to **(Name)** in the **Properties** window, you must use `chkFinished` in event procedures names, but you still have to use `CheckBox1` to return the control from the **Shapes** or **OLEObject** collection, as shown in the following example.

```
Private Sub chkFinished_Click()  
    ActiveSheet.OLEObjects("CheckBox1").Object.Value = 1  
End Sub
```



# Sheets Collection Object

Multiple objects [↳ Sheets](#)  
↳ Multiple objects

A collection of all the sheets in the specified or active workbook. The **Sheets** collection can contain [Chart](#) or [Worksheet](#) objects.

The **Sheets** collection is useful when you want to return sheets of any type. If you need to work with sheets of only one type, see the object topic for that sheet type.

## Using the Sheets Collection

Use the **Sheets** property to return the **Sheets** collection. The following example prints all sheets in the active workbook.

```
Sheets.PrintOut
```

Use the [Add](#) method to create a new sheet and add it to the collection. The following example adds two chart sheets to the active workbook, placing them after sheet two in the workbook.

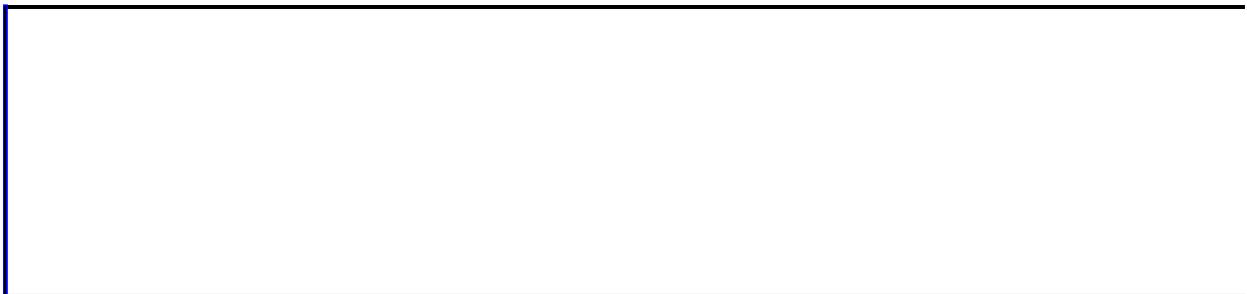
```
Sheets.Add type:=xlChart, count:=2, after:=Sheets(2)
```

Use **Sheets(index)**, where *index* is the sheet name or index number, to return a single **Chart** or **Worksheet** object. The following example activates the sheet named "sheet1."

```
Sheets("sheet1").Activate
```

Use **Sheets(array)** to specify more than one sheet. The following example moves the sheets named "Sheet4" and "Sheet5" to the beginning of the workbook.

```
Sheets(Array("Sheet4", "Sheet5")).Move before:=Sheets(1)
```



# SmartTagActions Collection

[SmartTag](#) └ [SmartTagActions](#)  
└ [SmartTagAction](#)

A collection of **SmartTagAction** objects that represent the actions that can be performed with smart tags.

## Using the SmartTagActions collection

Use the **SmartTagActions** property of the **SmartTag** object to return a **SmartTagActions** collection.

This example inserts a refreshable stock quote for the ticker symbol "MSFT" and it assumes the host system is connected to the Internet.

```
Sub ExecuteASmartTag()  
    Dim strAction As String  
  
    strAction = "Insert refreshable stock price"  
  
    ' Enable smart tags to be embedded and recognized.  
    ActiveWorkbook.SmartTagOptions.EmbedSmartTags = True  
    Application.SmartTagRecognizers.Recognize = True  
  
    ' Invoke a smart tag for the Microsoft ticker symbol.  
    With Range("A1")  
        .Formula = "MSFT"  
        .SmartTags( _  
            "urn:schemas-microsoft-com:office:smartsheet#stockticker"  
            .SmartTagActions(strAction).Execute  
        End With  
    End Sub
```



# SmartTagRecognizers Collection

[Application](#) └ [SmartTagRecognizers](#)  
└ [SmartTagRecognizer](#)

A collection of **SmartTagRecognizer** objects that represent recognition engines which label data with types of information as you work in Microsoft Excel.

## Using the SmartTagRecognizers collection

Use the **SmartTagRecognizers** property of the **Application** object to return a **SmartTagRecognizers** collection.

This example displays the first smart tag recognizer item available for the application or displays a message that none exist.

```
Sub CheckforSmartTagRecognizers()  
    ' Handle run-time error if no smart tag recognizers exist.  
    On Error Goto No_SmartTag_Recognizers_In_List  
  
    ' Notify the user of the first smart tag recognizer item.  
    MsgBox "The first smart tag recognizer is: " & _  
        Application.SmartTagRecognizers.Item(1)  
    Exit Sub  
  
No_SmartTag_Recognizers_In_List:  
    MsgBox "No smart tag recognizers exist in list."  
  
End Sub
```



# SmartTags Collection

Multiple objects [SmartTags](#)  
└ [SmartTag](#)  
└ Multiple objects

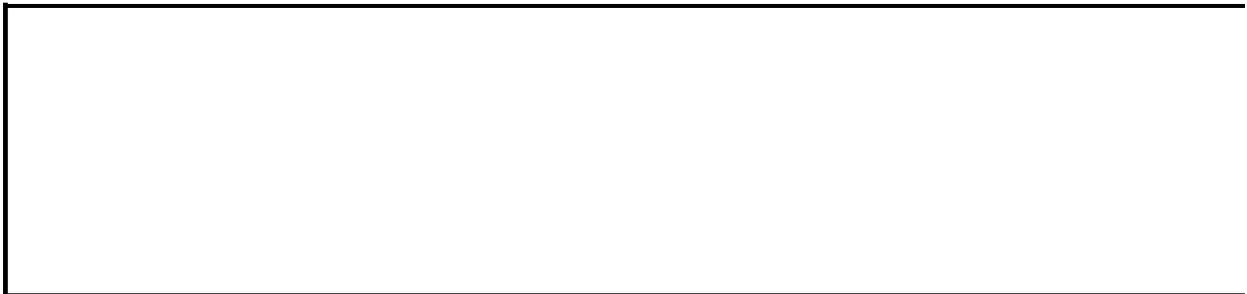
A collection of **SmartTag** objects that represent the identifiers assigned to each cell.

## Using the SmartTags collection

Use the **SmartTags** property of the **Range** collection or **Worksheet** object, to return a **SmartTag** collection. The following example demonstrates the use of the **SmartTags** property with the **Add** method.

This example adds a smart tag titled "MSFT" to cell A1, then adds extra metadata called "Market" with the value of "Nasdaq" to the smart tag and then returns the value of the property to the user. This example assumes the host system is connected to the Internet.

```
Sub UseProperties()  
  
    Dim strLink As String  
    Dim strType As String  
  
    ' Define smart tag variables.  
    strLink = "urn:schemas-microsoft-com:smarttags#StockTickerSymbol"  
    strType = "stockview"  
  
    ' Enable smart tags to be embedded and recognized.  
    ActiveWorkbook.SmartTagOptions.EmbedSmartTags = True  
    Application.SmartTagRecognizers.Recognize = True  
  
    Range("A1").Formula = "MSFT"  
  
    ' Add a property for MSFT smart tag and define it's value.  
    Range("A1").SmartTags.Add(strLink).Properties.Add _  
        Name:="Market", Value:="Nasdaq"  
  
    ' Notify the user of the smart tag's value.  
    MsgBox Range("A1").SmartTags.Add(strLink).Properties("Market").V  
  
End Sub
```



# Styles Collection

[Workbook](#) └ [Styles](#)  
└ [Style](#)  
└ Multiple objects

A collection of all the [Style](#) objects in the specified or active workbook. Each **Style** object represents a style description for a range. The **Style** object contains all style attributes (font, number format, alignment, and so on) as properties. There are several built-in styles— including Normal, Currency, and Percent — which are listed in the **Style name** box in the **Style** dialog box (**Format** menu).

## Using the Styles Collection

Use the **Styles** property to return the **Styles** collection. The following example creates a list of style names on worksheet one in the active workbook.

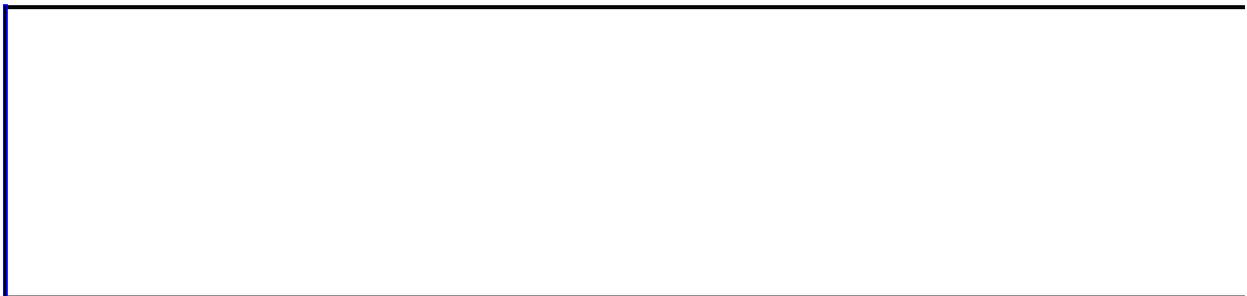
```
For i = 1 To ActiveWorkbook.Styles.Count
    Worksheets(1).Cells(i, 1) = ActiveWorkbook.Styles(i).Name
Next
```

Use the [Add](#) method to create a new style and add it to the collection. The following example creates a new style based on the Normal style, modifies the border and font, and then applies the new style to cells A25:A30.

```
With ActiveWorkbook.Styles.Add(Name:="Bookman Top Border")
    .Borders(xlTop).LineStyle = xlDouble
    .Font.Bold = True
    .Font.Name = "Bookman"
End With
Worksheets(1).Range("A25:A30").Style = "Bookman Top Border"
```

Use **Styles(index)**, where *index* is the style index number or name, to return a single **Style** object from the workbook **Styles** collection. The following example changes the Normal style for the active workbook by setting its **Bold** property.

```
ActiveWorkbook.Styles("Normal").Font.Bold = True
```



# Trendlines Collection Object

[Trendlines](#)  [Trendline](#)  
 Multiple objects

A collection of all the [Trendline](#) objects for the specified series. Each **Trendline** object represents a trendline in a chart. A trendline shows the trend, or direction, of data in a series.

## Using the Trendlines Collection

Use the **Trendlines** method to return the **Trendlines** collection. The following example displays the number of trendlines for series one in Chart1.

```
MsgBox Charts(1).SeriesCollection(1).Trendlines.Count
```

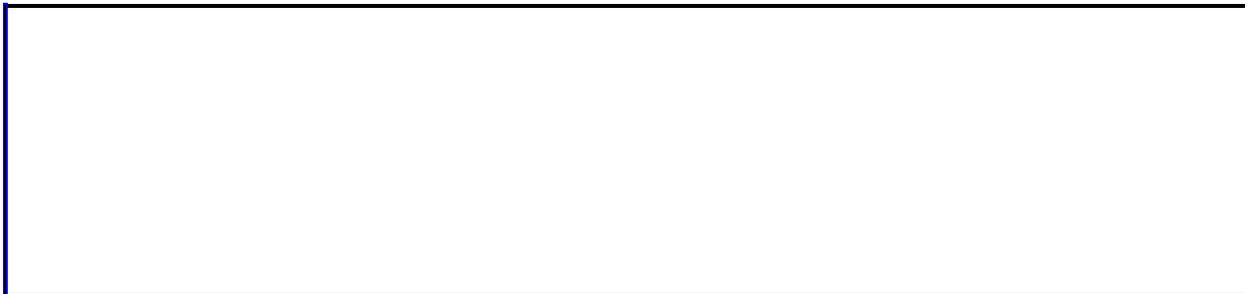
Use the [Add](#) method to create a new trendline and add it to the series. The following example adds a linear trendline to the first series in embedded chart one on Sheet1.

```
Worksheets("sheet1").ChartObjects(1).Chart.SeriesCollection(1) _  
    .Trendlines.Add type:=xlLinear, name:="Linear Trend"
```

Use **Trendlines(index)**, where *index* is the trendline index number, to return a single **TrendLine** object. The following example changes the trendline type for the first series in embedded chart one on worksheet one. If the series has no trendline, this example will fail.

```
Worksheets(1).ChartObjects(1).Chart. _  
    SeriesCollection(1).Trendlines(1).Type = xlMovingAvg
```

The index number denotes the order in which the trendlines were added to the series. `Trendlines(1)` is the first trendline added to the series, and `Trendlines(Trendlines.Count)` is the last one added.



# UsedObjects Collection

[Application](#)  $\perp$  [UsedObjects](#)

Represents objects that have been allocated in a workbook.

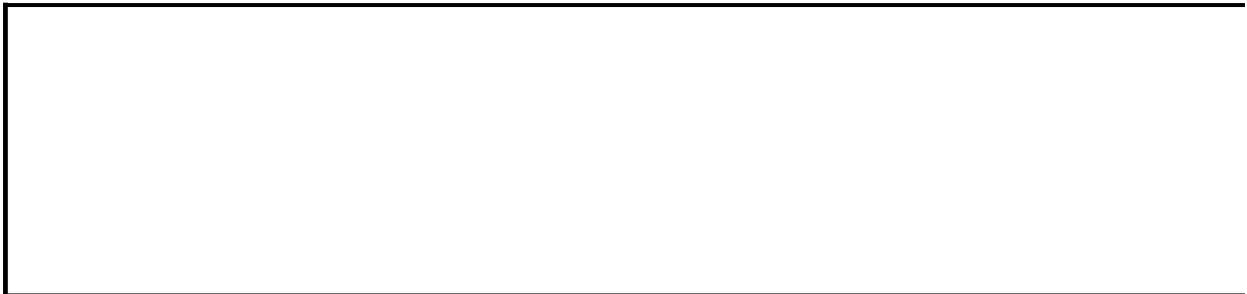
## Using the UsedObjects collection

Use the **UsedObjects** property of the **Application** object to return a **UsedObjects** collection.

Once a **UsedObjects** collection is returned, you can determine the quantity of used objects in a Microsoft Excel application using the **Count** property.

In this example, Microsoft Excel determines the quantity of objects that have been allocated and notifies the user. This example assumes a recalculation was performed in the application and was interrupted before finishing.

```
Sub CountUsedObjects()  
    MsgBox "The number of used objects in this application is: " & _  
        Application.UsedObjects.Count  
End Sub
```



# UserAccessList Collection

[AllowEditRange](#)  [UserAccessList](#)  
 [UserAccess](#)

A collection of **UserAccess** objects that represent the user access for protected ranges.

## Using the **UserAccessList** Collection

Use the **Users** property of the **ProtectedRange** object to return a **UserAccessList** collection.

Once a **UserAccessList** collection is returned you can use the **Count** property to determine the number of users that have access to a protected range. In the following example, Microsoft Excel notifies the user the numbers users that have access to the first protected range. This example assumes that a protected range exists on the active worksheet.

```
Sub UseDeleteAll()  
    Dim wksSheet As Worksheet  
    Set wksSheet = Application.ActiveSheet  
    ' Notify the user the number of users that can access the protec  
    MsgBox wksSheet.Protection.AllowEditRanges(1).Users.Count  
End Sub
```



# VPageBreaks Collection Object

Multiple objects [└ VPageBreaks](#)

[└ VPageBreak](#)

[└ Multiple objects](#)

A collection of vertical page breaks within the print area. Each vertical page break is represented by a [VPageBreak](#) object.

## Using the VPageBreaks Collection

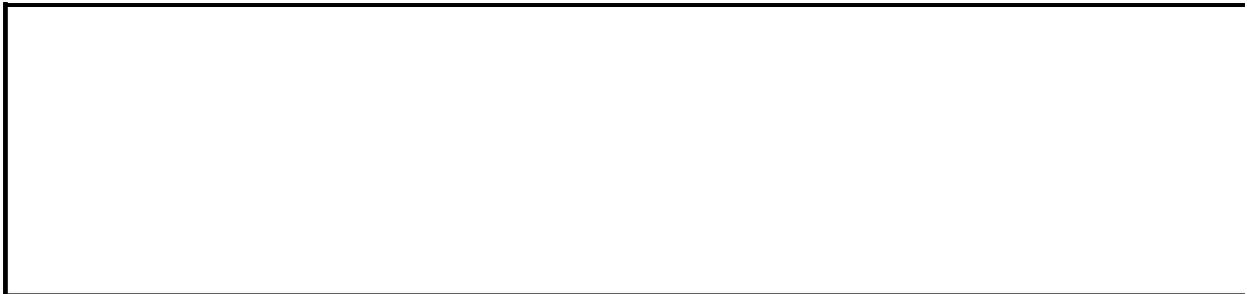
Use the [VPageBreaks](#) property to return the **VPageBreaks** collection. Use the [Add](#) method to add a vertical page break. The following example adds a vertical page break to the left of the active cell.

```
ActiveSheet.VPageBreaks.Add Before:=ActiveCell
```

If you add a page break that does not intersect the print area, then the newly-added **VPageBreak** object will not appear in the **VPageBreaks** collection for the print area. The contents of the collection may change if the print area is resized or redefined.

When the [Application](#) property, [Count](#) property, [Creator](#) property, [Item](#) property, [Parent](#) property or [Add](#) method is used in conjunction with the **VPageBreaks** property:

- For an automatic print area, the **VPageBreaks** property applies only to the page breaks within the print area.
- For a user-defined print area of the same range, the **VPageBreaks** property applies to all of the page breaks.



# Watches Collection

[Application](#) └ [Watches](#)  
└ [Watch](#)

A collection of all the [Watch](#) objects in a specified application.

## Using the Watches collection

Use the **Watches** property of the **Application** object to return a **Watches** collection.

In the following example, Microsoft Excel creates a new **Watch** object using the **Add** method. This example creates a summation formula in cell A3, and then adds this cell to the watch facility.

```
Sub AddWatch()  
  
    With Application  
        .Range("A1").Formula = 1  
        .Range("A2").Formula = 2  
        .Range("A3").Formula = "=Sum(A1:A2)"  
        .Range("A3").Select  
        .Watches.Add Source:=ActiveCell  
    End With  
  
End Sub
```

You can specify to remove individual cells from the watch facility by using the **Delete** method of the **Watches** collection. This example deletes cell A3 on worksheet 1 of book 1 from the Watch Window. This example assumes you have added the cell A3 on sheet 1 of book 1 (using the previous example to add a **Watch** object).

```
Sub DeleteAWatch()  
  
    Application.Watches(Workbooks("Book1").Sheets("Sheet1").Range("A3").Delete  
  
End Sub
```

You can also specify to remove all cells from the Watch Window, by using the **Delete** method of the **Watches** collection. This example deletes all cells from the Watch Window.

```
Sub DeleteAllWatches()  
  
    Application.Watches.Delete
```

End Sub



# Windows Collection Object

Multiple objects [└ Windows](#)  
└ [Window](#)  
└ Multiple objects

A collection of all the [Window](#) objects in Microsoft Excel. The **Windows** collection for the **Application** object contains all the windows in the application, whereas the **Windows** collection for the **Workbook** object contains only the windows in the specified workbook.

## Using the Windows Collection

Use the **Windows** property to return the **Windows** collection. The following example cascades all the windows that are currently displayed in Microsoft Excel.

```
Windows.Arrange arrangeStyle:=xlCascade
```

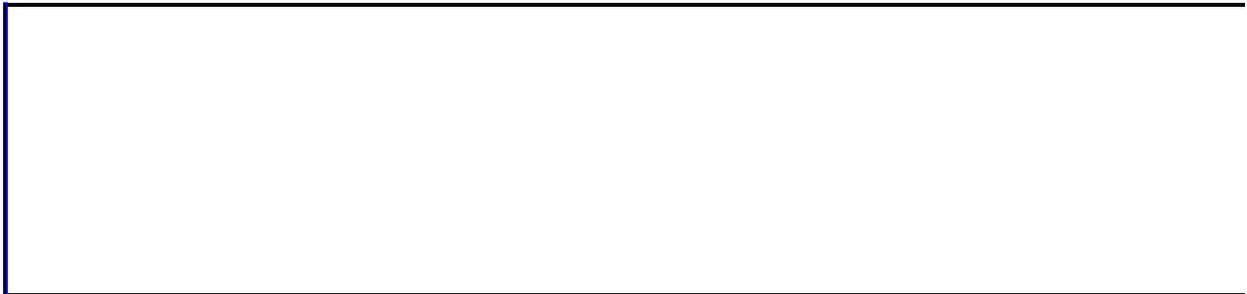
Use the [NewWindow](#) method to create a new window and add it to the collection. The following example creates a new window for the active workbook.

```
ActiveWorkbook.NewWindow
```

Use **Windows(index)**, where *index* is the window name or index number, to return a single **Window** object. The following example maximizes the active window.

```
Windows(1).WindowState = xlMaximized
```

Note that the active window is always `windows(1)`.



# Workbooks Collection

[Application](#) └ [Workbooks](#)  
└ [Workbook](#)  
└ Multiple objects

A collection of all the [Workbook](#) objects that are currently open in the Microsoft Excel application.

## Using the Workbooks Collection

Use the [Workbooks](#) property to return the **Workbooks** collection. The following example closes all open workbooks.

```
Workbooks.Close
```

Use the [Add](#) method to create a new, empty workbook and add it to the collection. The following example adds a new, empty workbook to Microsoft Excel.

```
Workbooks.Add
```

Use the [Open](#) method to open a file. This creates a new workbook for the opened file. The following example opens the file Array.xls as a read-only workbook.

```
Workbooks.Open FileName:="Array.xls", ReadOnly:=True
```

For more information about using a single **Workbook** object, see the [Workbook](#) object.



# Worksheets Collection

[Worksheets](#) └ Multiple objects

A collection of all the [Worksheet](#) objects in the specified or active workbook. Each **Worksheet** object represents a worksheet.

## Using the Worksheets Collection

Use the **Worksheets** property to return the **Worksheets** collection. The following example moves all the worksheets to the end of the workbook.

```
Worksheets.Move After:=Sheets(Sheets.Count)
```

Use the [Add](#) method to create a new worksheet and add it to the collection. The following example adds two new worksheets before sheet one of the active workbook.

```
Worksheets.Add Count:=2, Before:=Sheets(1)
```

Use **Worksheets(index)**, where *index* is the worksheet index number or name, to return a single **Worksheet** object. The following example hides worksheet one in the active workbook.

```
Worksheets(1).Visible = False
```

The **Worksheet** object is also a member of the [Sheets](#) collection. The **Sheets** collection contains all the sheets in the workbook (both chart sheets and worksheets).



# XmlMaps Collection

[Workbook](#) └ [XmlMaps](#)  
└ [XmlMap](#)  
└ Multiple objects

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Represents the collection of **XmlMap** objects that have been added to a workbook.

## Using the XmlMaps Collection

Use the **Add** method to add an XML map to a workbook.

```
Sub AddXmlMap()  
    Dim strSchemaLocation As String  
  
    strSchemaLocation = "http://example.microsoft.com/schemas/Custom"  
    ActiveWorkbook.XmlMaps.Add strSchemaLocation, "Root"  
End Sub
```



# XmlNamespaces Collection

[Workbook](#)  [XmlNamespaces](#)  
 [XmlNamespace](#)

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Represents the collection of **XmlNamespace** objects in a workbook.

## Using the XmlNamespaces Collection

Use the **Item** method to access a particular **XmlNamespace** object.

Use the **Value** property to return a **String** that lists the namespaces that have been added to a workbook.



# XmlSchemas Collection

[XmlMap](#) └ [XmlSchemas](#)  
└ [XmlSchema](#)  
└ [XmlNamespace](#)

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Represents the collection of **XmlSchema** objects contained by an **XmlMap** object.

## Using the XmlSchemas Collection

Use the **Schemas** property of the **XmlMap** object to return the **XmlSchemas** collection.

Use the **Item** method to return an **XmlSchema** object from the **XmlSchemas** collection.



# AddIn Object

[Application](#) └ [AddIns](#)  
└ [AddIn](#)

Represents a single add-in, either installed or not installed. The **AddIn** object is a member of the [AddIns](#) collection. The **AddIns** collection contains a list of all the add-ins available to Microsoft Excel, regardless of whether they're installed. This list corresponds to the list of add-ins displayed in the **Add-Ins** dialog box (**Tools** menu).

## Using the Addin Object

Use **AddIns**(*index*), where *index* is the add-in title or index number, to return a single **AddIn** object. The following example installs the Analysis Toolpak add-in.

```
AddIns("analysis toolpak").Installed = True
```

Don't confuse the add-in title, which appears in the **Add-Ins** dialog box, with the add-in name, which is the file name of the add-in. You must spell the add-in title exactly as it's spelled in the **Add-Ins** dialog box, but the capitalization doesn't have to match.

The index number represents the position of the add-in in the **Add-ins available** box in the **Add-Ins** dialog box. The following example creates a list that contains specified properties of the available add-ins.

```
With Worksheets("sheet1")
    .Rows(1).Font.Bold = True
    .Range("a1:d1").Value = _
        Array("Name", "Full Name", "Title", "Installed")
    For i = 1 To AddIns.Count
        .Cells(i + 1, 1) = AddIns(i).Name
        .Cells(i + 1, 2) = AddIns(i).FullName
        .Cells(i + 1, 3) = AddIns(i).Title
        .Cells(i + 1, 4) = AddIns(i).Installed
    Next
    .Range("a1").CurrentRegion.Columns.AutoFit
End With
```

## Remarks

The **Add** method adds an add-in to the list of available add-ins but doesn't install the add-in. Set the **Installed** property of the add-in to **True** to install the add-in. To install an add-in that doesn't appear in the list of available add-ins, you must first use the **Add** method and then set the **Installed** property. This can be done in a single step, as shown in the following example (note that you use the name of the add-in, not its title, with the **Add** method).

```
AddIns.Add("generic.xll").Installed = True
```

Use **Workbooks**(*index*) where *index* is the add-in filename (not title) to return a reference to the workbook corresponding to a loaded add-in. You must use the file name because loaded add-ins don't normally appear in the **Workbooks** collection. This example sets the *wb* variable to the workbook for Myaddin.xla.

```
Set wb = Workbooks("myaddin.xla")
```

The following example sets the *wb* variable to the workbook for the Analysis Toolpak add-in.

```
Set wb = Workbooks(AddIns("analysis toolpak").Name)
```

If the **Installed** property returns **True**, but calls to functions in the add-in still fail, the add-in may not actually be loaded. This is because the **Addin** object represents the existence and installed state of the add-in but doesn't represent the actual contents of the add-in workbook. To guarantee that an installed add-in is loaded, you should open the add-in workbook. The following example opens the workbook for the add-in named "My Addin" if the add-in isn't already present in the **Workbooks** collection.

```
On Error Resume Next      ' turn off error checking
Set wbMyAddin = Workbooks(Addins("My Addin").Name)
lastError = Err
On Error Goto 0           ' restore error checking
If lastError <> 0 Then
    ' the add-in workbook isn't currently open. Manually open it.
    Set wbMyAddin = Workbooks.Open(Addins("My Addin").FullName)
```

End If



# Adjustments Object

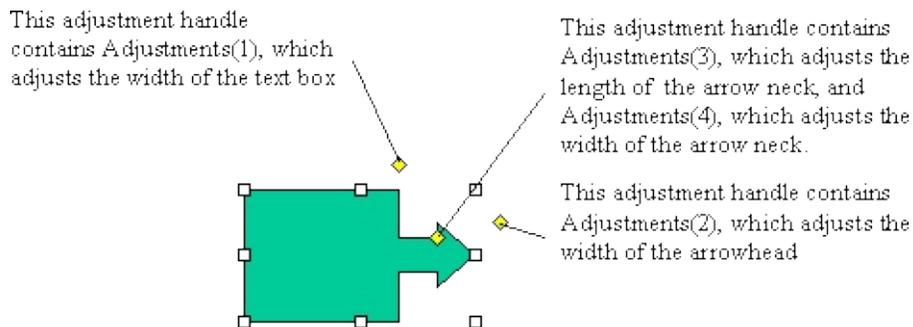
Multiple objects [Adjustments](#)

Contains a collection of adjustment values for the specified AutoShape, WordArt object, or connector. Each adjustment value represents one way an adjustment handle can be adjusted. Because some adjustment handles can be adjusted in two ways— for instance, some handles can be adjusted both horizontally and vertically— a shape can have more adjustment values than it has adjustment handles. A shape can have up to eight adjustments.

## Using the Adjustments Object

Use the **Adjustments** property to return an **Adjustments** object. Use **Adjustments(index)**, where *index* is the adjustment value's index number, to return a single adjustment value.

Different shapes have different numbers of adjustment values, different kinds of adjustments change the geometry of a shape in different ways, and different kinds of adjustments have different ranges of valid values. For example, the following illustration shows what each of the four adjustment values for a right-arrow callout contributes to the definition of the callout's geometry.



**Note** Because each adjustable shape has a different set of adjustments, the best way to verify the adjustment behavior for a specific shape is to manually create an instance of the shape, make adjustments with the macro recorder turned on, and then examine the recorded code.

The following table summarizes the ranges of valid adjustment values for different types of adjustments. In most cases, if you specify a value that's beyond the range of valid values, the closest valid value will be assigned to the adjustment.

Type of adjustment	Valid values
Linear (horizontal	Generally the value 0.0 represents the left or top edge of the shape and the value 1.0 represents the right or bottom edge of the shape. Valid values correspond to valid adjustments you can make to the shape manually. For example, if you can only pull an adjustment handle half way across the shape manually, the maximum value for

or vertical) the corresponding adjustment will be 0.5. For shapes such as connectors and callouts, where the values 0.0 and 1.0 represent the limits of the rectangle defined by the starting and ending points of the connector or callout line, negative numbers and numbers greater than 1.0 are valid values.

**Radial** An adjustment value of 1.0 corresponds to the width of the shape. The maximum value is 0.5, or half way across the shape.

**Angle** Values are expressed in degrees. If you specify a value outside the range – 180 to 180, it will be normalized to be within that range.

The following example adds a right-arrow callout to myDocument and sets adjustment values for the callout. Note that although the shape has only three adjustment handles, it has four adjustments. Adjustments three and four both correspond to the handle between the head and neck of the arrow.

```
Set myDocument = Worksheets(1)
Set rac = myDocument.Shapes.AddShape(msoShapeRightArrowCallout, _
    10, 10, 250, 190)
With rac.Adjustments
    .Item(1) = 0.5    'adjusts width of text box
    .Item(2) = 0.15  'adjusts width of arrow head
    .Item(3) = 0.8   'adjusts length of arrow head
    .Item(4) = 0.4   'adjusts width of arrow neck
End With
```



# AllowEditRange Object

[Protection](#) └ [AllowEditRanges](#)  
└ [AllowEditRange](#)  
└ Multiple objects

Represents the cells that can be edited on a protected worksheet.

## Using the AllowEditRange Object

Use the [Add](#) method or the [Item](#) property of the [AllowEditRanges](#) collection to return an **AllowEditRange** object.

Once an **AllowEditRange** object has been returned, you can use the [ChangePassword](#) method to change the password to access a range that can be edited on a protected worksheet.

In this example, Microsoft Excel allows edits to range "A1:A4" on the active worksheet, notifies the user, then changes the password for this specified range and notifies the user of this change.

```
Sub UseChangePassword()  
  
    Dim wksOne As Worksheet  
    Dim wksPassword As String  
  
    Set wksOne = Application.ActiveSheet  
  
    wksPassword = InputBox ("Enter password for the worksheet")  
  
    ' Establish a range that can allow edits  
    ' on the protected worksheet.  
    wksOne.Protection.AllowEditRanges.Add _  
        Title:="Classified", _  
        Range:=Range("A1:A4"), _  
        Password:=wksPassword  
  
    MsgBox "Cells A1 to A4 can be edited on the protected worksheet."  
  
    ' Change the password.  
  
    wksPassword = InputBox ("Enter the new password for the workshee  
  
    wksOne.Protection.AllowEditRanges(1).ChangePassword _  
        Password:=wksPassword  
  
    MsgBox "The password for these cells has been changed."  
  
End Sub
```



# Application Object

[Application](#) └ Multiple objects

Represents the entire Microsoft Excel application. The **Application** object contains:

- Application-wide settings and options (many of the options in the **Options** dialog box (**Tools** menu), for example).
- Methods that return top-level objects, such as **ActiveCell**, **ActiveSheet**, and so on.

## Using the Application Object

Use the **Application** property to return the **Application** object. The following example applies the **Windows** property to the **Application** object.

```
Application.Windows("book1.xls").Activate
```

The following example creates a Microsoft Excel workbook object in another application and then opens a workbook in Microsoft Excel.

```
Set xl = CreateObject("Excel.Sheet")  
xl.Application.Workbooks.Open "newbook.xls"
```

## Remarks

Many of the properties and methods that return the most common user-interface objects, such as the active cell ([ActiveCell](#) property), can be used without the **Application** object qualifier. For example, instead of writing `Application.ActiveCell.Font.Bold = True`, you can write `ActiveCell.Font.Bold = True`.



# AutoCorrect Object

[Application](#) `└ AutoCorrect`

Contains Microsoft Excel AutoCorrect attributes (capitalization of names of days, correction of two initial capital letters, automatic correction list, and so on).

## Using the AutoCorrect Object

Use the [AutoCorrect](#) property to return the **AutoCorrect** object. The following example sets Microsoft Excel to correct words that begin with two initial capital letters.

```
With Application.AutoCorrect  
    .TwoInitialCapitals = True  
    .ReplaceText = True  
End With
```



# AutoFilter Object

[Worksheet](#) └ [AutoFilter](#)  
└ Multiple objects

Represents autofiltering for the specified worksheet.

## Using the AutoFilter Object

Use the **AutoFilter** property to return the **AutoFilter** object. Use the **Filters** method to return a collection of individual column filters. Use the **Range** method to return the **Range** object that represents the entire filtered range. The following example stores the address and filtering criteria for the current filtering and then applies new filters.

```
Dim w As Worksheet
Dim filterArray()
Dim currentFiltRange As String

Sub ChangeFilters()

Set w = Worksheets("Crew")
With w.AutoFilter
    currentFiltRange = .Range.Address
    With .Filters
        ReDim filterArray(1 To .Count, 1 To 3)
        For f = 1 To .Count
            With .Item(f)
                If .On Then
                    filterArray(f, 1) = .Criteria1
                    If .Operator Then
                        filterArray(f, 2) = .Operator
                        filterArray(f, 3) = .Criteria2
                    End If
                End If
            End With
        End With
    Next
End With

w.AutoFilterMode = False
w.Range("A1").AutoFilter field:=1, Criteria1:="S"

End Sub
```

To create an **AutoFilter** object for a worksheet, you must turn autofiltering on for a range on the worksheet either manually or using the **AutoFilter** method of the **Range** object. The following example uses the values stored in module-level variables in the previous example to restore the original autofiltering to the Crew worksheet.

```
Sub RestoreFilters()  
Set w = Worksheets("Crew")  
w.AutoFilterMode = False  
For col = 1 To UBound(filterArray(), 1)  
    If Not IsEmpty(filterArray(col, 1)) Then  
        If filterArray(col, 2) Then  
            w.Range(currentFiltRange).AutoFilter field:=col, _  
                Criteria1:=filterArray(col, 1), _  
                Operator:=filterArray(col, 2), _  
                Criteria2:=filterArray(col, 3)  
        Else  
            w.Range(currentFiltRange).AutoFilter field:=col, _  
                Criteria1:=filterArray(col, 1)  
        End If  
    End If  
Next  
End Sub
```



# AutoRecover Object

[Application](#) <sup>L</sup> [AutoRecover](#)

Represents the automatic recovery features of a workbook. Properties for the **AutoRecover** object determine the path and time interval for backing up all files.

## Using the **AutoRecover** object

Use the [AutoRecover](#) property of the [Application](#) object to return an **AutoRecover** object.

Use the [Path](#) property of the **AutoRecover** object to set the path for where the AutoRecover file will be saved. The following example sets the path of the AutoRecover file to drive C.

```
Sub SetPath()  
    Application.AutoRecover.Path = "C:\"  
End Sub
```

Use the [Time](#) property of the **AutoRecover** object to set the time interval for backing up all files.

**Note** Units for the **Time** property are in minutes.

```
Sub SetTime()  
    Application.AutoRecover.Time = 5  
End Sub
```



# Axis Object

[Axes](#) └ [Axis](#)  
└ Multiple objects

Represents a single axis in a chart. The **Axis** object is a member of the [Axes](#) collection.

## Using the Axis Object

Use **Axes**(*type*, *group*) where *type* is the axis type and *group* is the axis group to return a single **Axis** object. *Type* can be one of the following **XIAxisType** constants: **xlCategory**, **xlSeries**, or **xlValue**. *Group* can be one of the following **XIAxisGroup** constants: **xlPrimary** or **xlSecondary**. For more information, see the [Axes](#) method.

The following example sets the category axis title text on the chart sheet named "Chart1."

```
With Charts("chart1").Axes(xlCategory)
    .HasTitle = True
    .AxisTitle.Caption = "1994"
End With
```



# AxisTitle Object

[Axis](#) └ [AxisTitle](#)  
└ Multiple objects

Represents a chart axis title.

## Using the AxisTitle Object

Use the **AxisTitle** property to return an **AxisTitle** object. The following example activates embedded chart one, sets the value axis title text, sets the font to Bookman 10 point, and formats the word millions as italic.

```
Worksheets("sheet1").ChartObjects(1).Activate
With ActiveChart.Axes(xlValue)
    .HasTitle = True
    With .AxisTitle
        .Caption = "Revenue (millions)"
        .Font.Name = "bookman"
        .Font.Size = 10
        .Characters(10, 8).Font.Italic = True
    End With
End With
```

## Remarks

The **AxisTitle** object doesn't exist and cannot be used unless the [HasTitle](#) property for the axis is **True**.

---

---

---

# Border Object

Multiple objects [↳ Border](#)

Represents the border of an object.

## Using the Border Object

Most bordered objects (all except for the **Range** and **Style** objects) have a border that's treated as a single entity, regardless of how many sides it has. The entire border must be returned as a unit. Use the **Border** property to return the **Border** object for this kind of object. The following example activates the chart sheet named Chart1 places a dashed border around the chart area for the active chart and places a dotted border around the plot area.

```
Charts("chart1").Activate  
With ActiveChart  
    .ChartArea.Border.LineStyle = xlDash  
    .PlotArea.Border.LineStyle = xlDot  
End With
```

**Range** and **Style** objects have four discrete borders— left, right, top, and bottom — which can be returned individually or as a group. Use the **Borders** property to return the **Borders** collection, which contains all four borders and treats the borders as a unit. The following example adds a double border to cell A1 on worksheet one.

```
Worksheets(1).Range("A1").Borders.LineStyle = xlDouble
```

Use **Borders(index)**, where *index* identifies the border, to return a single **Border** object. The following example sets the color of the bottom border of cells A1:G1.

```
Worksheets("Sheet1").Range("A1:G1").  
    Borders(xlEdgeBottom).Color = RGB(255, 0, 0)
```

*Index* can be one of the following **XIBordersIndex** constants: **xlDiagonalDown**, **xlDiagonalUp**, **xlEdgeBottom**, **xlEdgeLeft**, **xlEdgeRight**, **xlEdgeTop**, **xlInsideHorizontal**, or **xlInsideVertical**.



[Show All](#)

# CalculatedMember Object

[PivotTable](#) └ [CalculatedMembers](#)  
└ [CalculatedMember](#)

Represents the calculated fields and calculated items for PivotTables with [Online Analytical Processing \(OLAP\)](#) data sources.

## Using the CalculatedMember object

Use the [Add](#) method or the [Item](#) property of the [CalculatedMembers](#) collection to return a **CalculatedMember** object.

With a **CalculatedMember** object you can check the validity of a calculated field or item in a PivotTable using the [IsValid](#) property.

**Note** The **IsValid** property will return **True** if the PivotTable is not currently connected to the data source. Use the [MakeConnection](#) method before testing the **IsValid** property.

The following example notifies the user if the calculated member is valid or not. This example assumes a PivotTable exists on the active worksheet that contains either a valid or invalid calculated member.

```
Sub CheckValidity()  
  
    Dim pvtTable As PivotTable  
    Dim pvtCache As PivotCache  
  
    Set pvtTable = ActiveSheet.PivotTables(1)  
    Set pvtCache = Application.ActiveWorkbook.PivotCaches.Item(1)  
  
    ' Handle run-time error if external source is not an OLEDB data  
    On Error GoTo Not_OLEDB  
  
    ' Check connection setting and make connection if necessary.  
    If pvtCache.IsConnected = False Then  
        pvtCache.MakeConnection  
    End If  
  
    ' Check if calculated member is valid.  
    If pvtTable.CalculatedMembers.Item(1).IsValid = True Then  
        MsgBox "The calculated member is valid."  
    Else  
        MsgBox "The calculated member is not valid."  
    End If  
  
End Sub
```



# CalloutFormat Object

Multiple objects [↳ CalloutFormat](#)

Contains properties and methods that apply to line callouts.

## Using the CalloutFormat Object

Use the **Callout** property to return a **CalloutFormat** object. The following example specifies the following attributes of shape three (a line callout) on myDocument: the callout will have a vertical accent bar that separates the text from the callout line; the angle between the callout line and the side of the callout text box will be 30 degrees; there will be no border around the callout text; the callout line will be attached to the top of the callout text box; and the callout line will contain two segments. For this example to work, shape three must be a callout.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(3).Callout
    .Accent = True
    .Angle = msoCalloutAngle30
    .Border = False
    .PresetDrop msoCalloutDropTop
    .Type = msoCalloutThree
End With
```



# CellFormat Object

[Application](#) └ [CellFormat](#)  
└ Multiple objects

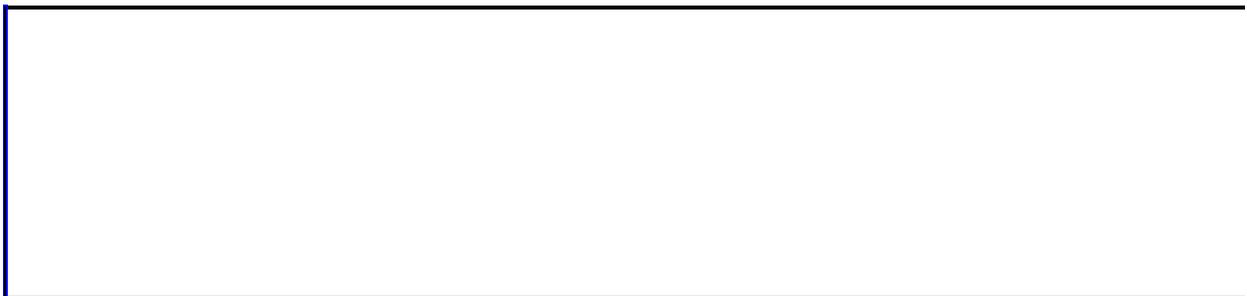
Represents the search criteria for the cell format.

## Using the CellFormat object

Use the **FindFormat** or **ReplaceFormat** properties of the **Application** object to return a **CellFormat** object.

With a **CellFormat** object, you can use the [Borders](#), [Font](#), or [Interior](#) properties of the **CellFormat** object, to define the search criteria for the cell format. The following example sets the search criteria for the interior of the cell format. In this scenario, the interior of cell A1 is set to yellow, which is then found and replaced with a green interior.

```
Sub ChangeCellFormat()  
  
    ' Set the interior of cell A1 to yellow.  
    Range("A1").Select  
    Selection.Interior.ColorIndex = 36  
    MsgBox "The cell format for cell A1 is a yellow interior."  
  
    ' Set the CellFormat object to replace yellow with green.  
    With Application  
        .FindFormat.Interior.ColorIndex = 36  
        .ReplaceFormat.Interior.ColorIndex = 35  
    End With  
  
    ' Find and replace cell A1's yellow interior with green.  
    ActiveCell.Replace What:="", Replacement:="", LookAt:=xlPart, _  
        SearchOrder:=xlByRows, MatchCase:=False, SearchFormat:=True,  
        ReplaceFormat:=True  
    MsgBox "The cell format for cell A1 is replaced with a green int  
  
End Sub
```



# Characters Object

Multiple objects [└ Characters](#)  
[└ Font](#)

Represents characters in an object that contains text. The **Characters** object lets you modify any sequence of characters contained in the full text string.

## Using the Characters Object

Use **Characters**(*start*, *length*), where *start* is the start character number and *length* is the number of characters, to return a **Characters** object. The following example adds text to cell B1 and then makes the second word bold.

```
With Worksheets("Sheet1").Range("B1")  
    .Value = "New Title"  
    .Characters(5, 5).Font.Bold = True  
End With
```

## Remarks

The **Characters** method is necessary only when you need to change some of an object's text without affecting the rest (you cannot use the **Characters** method to format a portion of the text if the object doesn't support rich text). To change all the text at the same time, you can usually apply the appropriate method or property directly to the object. The following example formats the contents of cell A5 as italic.

```
Worksheets("Sheet1").Range("A5").Font.Italic = True
```



# Chart Object

Multiple objects [↳ Chart](#)  
↳ Multiple objects

Represents a chart in a workbook. The chart can be either an embedded chart (contained in a [ChartObject](#) ) or a separate chart sheet.

## Using the Chart Object

The following properties and methods for returning a **Chart** object are described in this section:

- **Chart** property
- **Charts** method
- **ActiveChart** property
- **ActiveSheet** property

## Chart Property

Use the **Chart** property to return a **Chart** object that represents the chart contained in a **ChartObject** object. The following example sets the pattern for the chart area in embedded chart one on the worksheet named "Sheet1."

```
Worksheets("Sheet1").ChartObjects(1).Chart. _  
    ChartArea.Interior.Pattern = xlLightDown
```

## Charts Method

The [Charts](#) collection contains a **Chart** object for each chart sheet in a workbook. Use **Charts(index)**, where *index* is the chart-sheet index number or name, to return a single **Chart** object. The following example changes the color of series one on chart sheet one.

```
Charts(1).SeriesCollection(1).Interior.Color = RGB(255, 0, 0)
```

The chart index number represents the position of the chart sheet on the workbook tab bar. **Charts(1)** is the first (leftmost) chart in the workbook; **Charts(Charts.Count)** is the last (rightmost). All chart sheets are included in the index count, even if they're hidden. The chart-sheet name is shown on the workbook tab for the chart. You can use the [Name](#) property to set or return the chart name.

The following example moves the chart named Sales to the end of the active workbook.

```
Charts("Sales").Move after:=Sheets(Sheets.Count)
```

The **Chart** object is also a member of the [Sheets](#) collection. The **Sheets** collection contains all the sheets in the workbook (both chart sheets and worksheets). Use **Sheets(index)**, where *index* is the sheet index number or name, to return a single sheet.

## ActiveChart Property

When a chart is the active object, you can use the **ActiveChart** property to refer to it. A chart sheet is active if the user has selected it or it's been activated with the [Activate](#) method. The following example activates chart sheet one and then sets the chart type and title.

```
Charts(1).Activate
With ActiveChart
    .Type = xlLine
    .HasTitle = True
    .ChartTitle.Text = "January Sales"
End With
```

An embedded chart is active if the user has selected it or the [ChartObject](#) object that it's contained in has been activated with the **Activate** method. The following example activates embedded chart one on worksheet one and then sets the chart type and title. Notice that after the embedded chart has been activated, the code in this example is the same as that in the previous example. Using the **ActiveChart** property allows you to write Visual Basic code that can refer to either an embedded chart or a chart sheet (whichever is active).

```
Worksheets(1).ChartObjects(1).Activate
ActiveChart.Type = xlLine
ActiveChart.HasTitle = True
ActiveChart.ChartTitle.Text = "January Sales"
```

## ActiveSheet Property

When a chart sheet is the active sheet, you can use the **ActiveSheet** property to refer to it. The following example uses the **Activate** method to activate the chart sheet named Chart1 and then sets the interior color for series one in the chart to blue.

```
Charts("chart1").Activate  
ActiveSheet.SeriesCollection(1).Interior.ColorIndex = 5
```



# ChartArea Object

[Chart](#) └ [ChartArea](#)  
└ Multiple objects

Represents the chart area of a chart. The chart area on a 2-D chart contains the axes, the chart title, the axis titles, and the legend. The chart area on a 3-D chart contains the chart title and the legend; it doesn't include the plot area (the area within the chart area where the data is plotted). For information about formatting the plot area, see the [PlotArea](#) object.

## Using the ChartArea Object

Use the **ChartArea** property to return the **ChartArea** object. The following example sets the pattern for the chart area in embedded chart one on the worksheet named "Sheet1."

```
Worksheets("sheet1").ChartObjects(1).Chart. _  
    ChartArea.Interior.Pattern = xlLightDown
```



# ChartColorFormat Object

[ChartFillFormat](#)  [ChartColorFormat](#)

Used only with charts. Represents the color of a one-color object or the foreground or background color of an object with a gradient or patterned fill.

# Using the ChartColorFormat Object

Use one of the properties listed in the following table to return a **ChartColorFormat** object.

<b>To return a ChartColorFormat object that represents this</b>	<b>Use this property</b>	<b>With this object</b>
Background fill color (used in a shaded or patterned fill)	<a href="#"><u>BackColor</u></a>	<b>ChartFillFormat</b>
Foreground fill color (or just the fill color for a solid fill)	<a href="#"><u>ForeColor</u></a>	<b>ChartFillFormat</b>



# ChartFillFormat Object

Multiple objects [↳ ChartFillFormat](#)  
[↳ ChartColorFormat](#)

Used only with charts. Represents fill formatting for chart elements.

## Using the ChartFillFormat Object

Use the [Fill](#) property to return a **ChartFillFormat** object. The following example sets the foreground color, background color, and gradient for the chart area fill on chart one.

```
With Charts(1).ChartArea.Fill
    .Visible = True
    .ForeColor.SchemeColor = 15
    .BackColor.SchemeColor = 17
    .TwoColorGradient Style:=msoGradientHorizontal, Variant:=1
End With
```



# ChartGroup Object

[ChartGroups](#)   └─[ChartGroup](#)  
└─Multiple objects

Represents one or more series plotted in a chart with the same format. A chart contains one or more chart groups, each chart group contains one or more [series](#), and each series contains one or more [points](#). For example, a single chart might contain both a line chart group, containing all the series plotted with the line chart format, and a bar chart group, containing all the series plotted with the bar chart format. The **ChartGroup** object is a member of the [ChartGroups](#) collection.

## Using the ChartGroup Object

Use **ChartGroups**(*index*), where *index* is the chart-group index number, to return a single **ChartGroup** object. The following example adds drop lines to chart group one on chart sheet one.

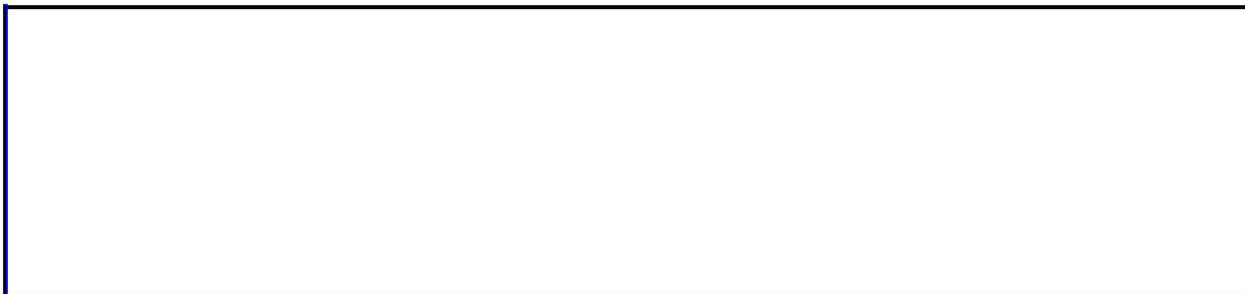
```
Charts(1).ChartGroups(1).HasDropLines = True
```

If the chart has been activated, you can use the **ActiveChart** property.

```
Charts(1).Activate  
ActiveChart.ChartGroups(1).HasDropLines = True
```

Because the index number for a particular chart group can change if the chart format used for that group is changed, it may be easier to use one of the named chart group shortcut methods to return a particular chart group. The **PieGroups** method returns the collection of pie chart groups in a chart, the **LineGroups** method returns the collection of line chart groups, and so on. Each of these methods can be used with an index number to return a single **ChartGroup** object, or without an index number to return a **ChartGroups** collection. The following chart group methods are available:

- [AreaGroups](#) method
- [BarGroups](#) method
- [ColumnGroups](#) method
- [DoughnutGroups](#) method
- [LineGroups](#) method
- [PieGroups](#) method



# ChartObject Object

[ChartObjects](#) └ [ChartObject](#)  
└ Multiple objects

Represents an embedded chart on a worksheet. The **ChartObject** object acts as a container for a [Chart](#) object. Properties and methods for the **ChartObject** object control the appearance and size of the embedded chart on the worksheet. The **ChartObject** object is a member of the [ChartObjects](#) collection. The **ChartObjects** collection contains all the embedded charts on a single sheet.

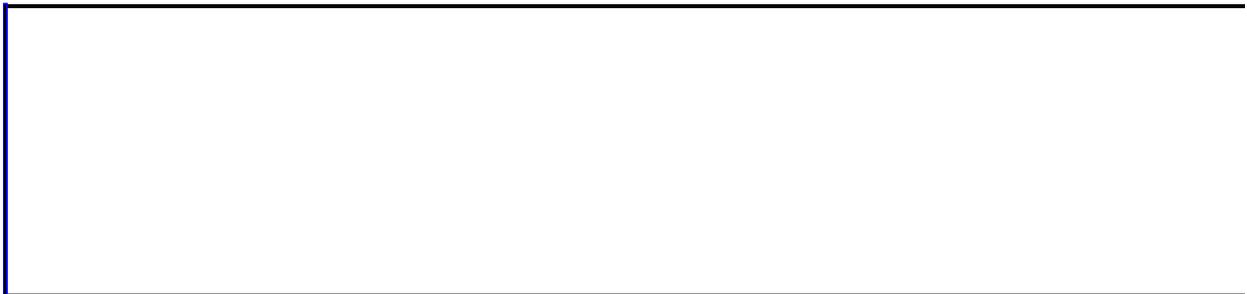
## Using the ChartObject Object

Use **ChartObjects**(*index*), where *index* is the embedded chart index number or name, to return a single **ChartObject** object. The following example sets the pattern for the chart area in embedded chart one on the worksheet named "Sheet1."

```
Worksheets("Sheet1").ChartObjects(1).Chart. _  
    ChartArea.Interior.Pattern = xlLightDown
```

The embedded chart name is shown in the Name box when the embedded chart is selected. Use the [Name](#) property to set or return the name of the **ChartObject** object. The following example puts rounded corners on the embedded chart named "Chart 1" on the worksheet named "Sheet1."

```
Worksheets("sheet1").ChartObjects("chart 1").RoundedCorners = True
```



# ChartTitle Object

[Chart](#) └ [ChartTitle](#)  
└ Multiple objects

Represents the chart title.

## Using the ChartTitle Object

Use the **ChartTitle** property to return the **ChartTitle** object. The following example adds a title to embedded chart one on the worksheet named "Sheet1."

```
With Worksheets("sheet1").ChartObjects(1).Chart
    .HasTitle = True
    .ChartTitle.Text = "February Sales"
End With
```

## Remarks

The **ChartTitle** object doesn't exist and cannot be used unless the [HasTitle](#) property for the chart is **True**.



# ColorFormat Object

Multiple objects [ColorFormat](#)

Represents the color of a one-color object, the foreground or background color of an object with a gradient or patterned fill, or the pointer color. You can set colors to an explicit red-green-blue value (by using the [RGB](#) property) or to a color in the color scheme (by using the [SchemeColor](#) property).

## Using the ColorFormat Object

Use one of the properties listed in the following table to return a **ColorFormat** object.

Use this property	With this object	To return a ColorFormat object that represents this
<a href="#">BackColor</a>	<b>FillFormat</b>	The background fill color (used in a shaded or patterned fill)
<a href="#">ForeColor</a>	<b>FillFormat</b>	The foreground fill color (or simply the fill color for a solid fill)
<a href="#">BackColor</a>	<b>LineFormat</b>	The background line color (used in a patterned line)
<a href="#">ForeColor</a>	<b>LineFormat</b>	The foreground line color (or just the line color for a solid line)
<a href="#">ForeColor</a>	<b>ShadowFormat</b>	The shadow color
<a href="#">ExtrusionColor</a>	<b>ThreeDFormat</b>	The color of the sides of an extruded object

Use the [RGB](#) property to set a color to an explicit red-green-blue value. The following example adds a rectangle to myDocument and then sets the foreground color, background color, and gradient for the rectangle's fill.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddShape(msoShapeRectangle, _
    90, 90, 90, 50).Fill
    .ForeColor.RGB = RGB(128, 0, 0)
    .BackColor.RGB = RGB(170, 170, 170)
    .TwoColorGradient msoGradientHorizontal, 1
End With
```



# Comment Object

[Worksheet](#) └ [Comments](#)  
└ [Comment](#)  
└ [Shape](#)

Represents a cell comment. The **Comment** object is a member of the [Comments](#) collection.

## Using the Comment Object

Use the [Comment](#) property to return a **Comment** object. The following example changes the text in the comment in cell E5.

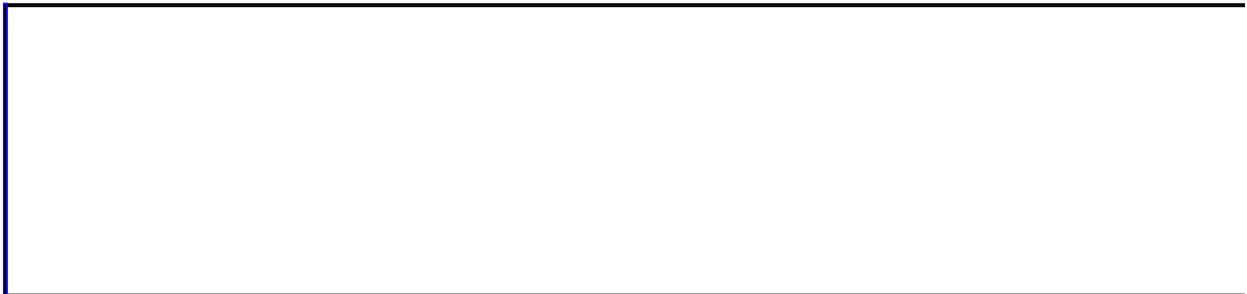
```
Worksheets(1).Range("E5").Comment.Text "reviewed on " & Date
```

Use **Comments**(*index*), where *index* is the comment number, to return a single comment from the **Comments** collection. The following example hides comment two on worksheet one.

```
Worksheets(1).Comments(2).Visible = False
```

Use the [AddComment](#) method to add a comment to a range. The following example adds a comment to cell E5 on worksheet one.

```
With Worksheets(1).Range("e5").AddComment  
    .Visible = False  
    .Text "reviewed on " & Date  
End With
```



# ConnectorFormat Object

Multiple objects [↳ ConnectorFormat](#)  
[↳ Shape](#)

Contains properties and methods that apply to connectors. A connector is a line that attaches two other shapes at points called connection sites. If you rearrange shapes that are connected, the geometry of the connector will be automatically adjusted so that the shapes remain connected.

## Using the ConnectorFormat Object

Use the **ConnectorFormat** property to return a **ConnectorFormat** object. Use the [BeginConnect](#) and [EndConnect](#) methods to attach the ends of the connector to other shapes in the document. Use the [RerouteConnections](#) method to automatically find the shortest path between the two shapes connected by the connector. Use the [Connector](#) property to see whether a shape is a connector.

Note that you assign a size and a position when you add a connector to the **Shapes** collection, but the size and position are automatically adjusted when you attach the beginning and end of the connector to other shapes in the collection. Therefore, if you intend to attach a connector to other shapes, the initial size and position you specify are irrelevant. Likewise, you specify which connection sites on a shape to attach the connector to when you attach the connector, but using the **RerouteConnections** method after the connector is attached may change which connection sites the connector attaches to, making your original choice of connection sites irrelevant.

The following example adds two rectangles to myDocument and connects them with a curved connector.

```
Set myDocument = Worksheets(1)
Set s = myDocument.Shapes
Set firstRect = s.AddShape(msoShapeRectangle, 100, 50, 200, 100)
Set secondRect = s.AddShape(msoShapeRectangle, 300, 300, 200, 100)
Set c = s.AddConnector(msoConnectorCurve, 0, 0, 0, 0)
With c.ConnectorFormat
    .BeginConnect ConnectedShape:=firstRect, ConnectionSite:=1
    .EndConnect ConnectedShape:=secondRect, ConnectionSite:=1
    c.RerouteConnections
End With
```

## Remarks

Connection sites are generally numbered according to the rules presented in the following table.

<b>Shape type</b>	<b>Connection site numbering scheme</b>
AutoShapes, WordArt, pictures, and OLE objects	The connection sites are numbered starting at the top and proceeding counterclockwise.
Freeforms	The connection sites are the vertices, and they correspond to the vertex numbers.

To figure out which number corresponds to which connection site on a complex shape, you can experiment with the shape while the macro recorder is turned on and then examine the recorded code; or you can create a shape, select it, and then run the following example. This code will number each connection site and attach a connector to it.

```
Set mainshape = ActiveWindow.Selection.ShapeRange(1)
With mainshape
    bx = .Left + .Width + 50
    by = .Top + .Height + 50
End With
With ActiveSheet
    For j = 1 To mainshape.ConnectionSiteCount
        With .Shapes.AddConnector(msoConnectorStraight, _
            bx, by, bx + 50, by + 50)
            .ConnectorFormat.EndConnect mainshape, j
            .ConnectorFormat.Type = msoConnectorElbow
            .Line.ForeColor.RGB = RGB(255, 0, 0)
            l = .Left
            t = .Top
        End With
        With .Shapes.AddTextbox(msoTextOrientationHorizontal, _
            l, t, 36, 14)
            .Fill.Visible = False
            .Line.Visible = False
            .TextFrame.Characters.Text = j
        End With
    Next j
End With
```



[Show All](#)

# ControlFormat Object

[Shape](#) └ [ControlFormat](#)

Contains [Microsoft Excel control](#) properties.

## Using the ControlFormat Object

Use the [ControlFormat](#) property to return a **ControlFormat** object. The following example sets the fill range for a list box control on worksheet one.

```
Worksheets(1).Shapes(1).ControlFormat.ListFillRange = "A1:A10"
```

If the shape isn't a control, the **ControlFormat** property fails; and if the control isn't a list box, the **ListFillRange** property fails.



# Corners Object

[Chart](#)  $\perp$  [Corners](#)

Represents the corners of a 3-D chart. This object isn't a collection.

## Using the Corners Object

Use the [Corners](#) property to return the **Corners** object. The following example selects the corners of chart one.

```
Charts(1).Corners.Select
```

If the chart isn't a 3-D chart, the **Corners** property fails.



[Show All](#)

# CubeField Object

Multiple objects [CubeField](#)  
└─ Multiple objects

Represents a hierarchy or measure field from an [OLAP cube](#). In a PivotTable report, the **CubeField** object is a member of the [CubeFields](#) collection.

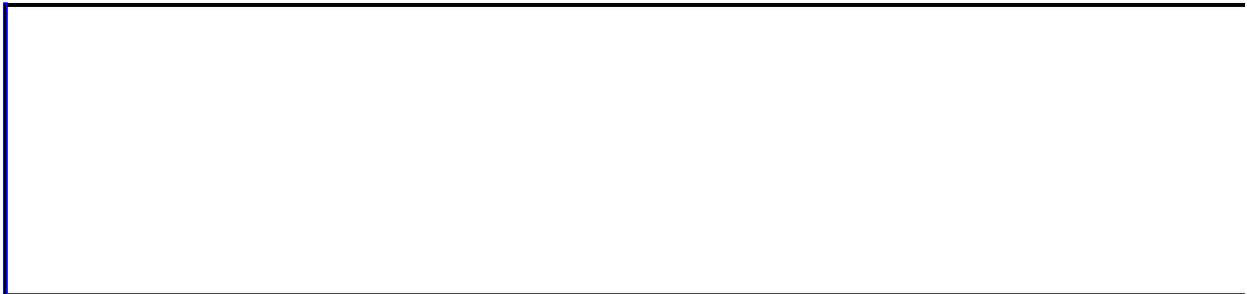
## Using the CubeField Object

Use the [CubeField](#) property to return the **CubeField** object. This example creates a list of the cube field names for all the hierarchy fields in the first OLAP-based PivotTable report on Sheet1.

```
Set objNewSheet = Worksheets.Add
objNewSheet.Activate
intRow = 1
For Each objPF in _
    Worksheets("Sheet1").PivotTables(1).PivotFields
    If objPF.CubeField.CubeFieldType = xlHierarchy Then
        objNewSheet.Cells(intRow, 1).Value = objPF.Name
        intRow = intRow + 1
    End If
Next objPF
```

Use **CubeFields(index)**, where *index* is the cube field's index number, to return a single **CubeField** object. The following example determines the name of the second cube field in the first PivotTable report on the active worksheet.

```
strAlphaName = _
    ActiveSheet.PivotTables(1).CubeFields(2).Name
```



# CustomProperty Object

Multiple objects [↳ CustomProperties](#)  
[↳ CustomProperty](#)

Represents identifier information. Identifier information can be used as metadata for XML.

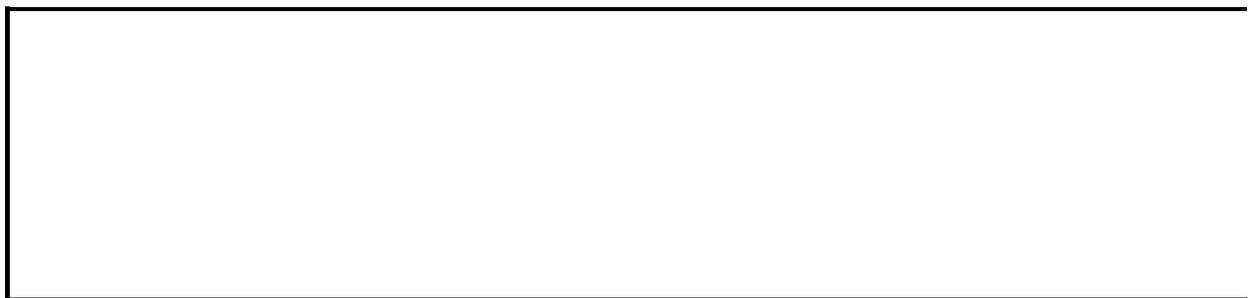
## Using the CustomProperty object

Use the **Add** method or the **Item** property of the **CustomProperties** collection to return a **CustomProperty** object.

Once a **CustomProperty** object is returned, you can add metadata to worksheets using the **CustomProperties** property with the **Add** method.

The following example demonstrates this feature. In this example, Microsoft Excel adds identifier information to the active worksheet and returns the name and value to the user.

```
Sub CheckCustomProperties()  
    Dim wksSheet1 As Worksheet  
    Set wksSheet1 = Application.ActiveSheet  
    ' Add metadata to worksheet.  
    wksSheet1.CustomProperties.Add _  
        Name:="Market", Value:="Nasdaq"  
    ' Display metadata.  
    With wksSheet1.CustomProperties.Item(1)  
        MsgBox .Name & vbTab & .Value  
    End With  
End Sub
```



# CustomView Object

[CustomViews](#) | [CustomView](#)

Represents a custom workbook view. The **CustomView** object is a member of the [CustomViews](#) collection.

## Using the CustomView Object

Use **CustomViews**(*index*), where *index* is the name or index number of the custom view, to return a **CustomView** object. The following example shows the custom view named "Current Inventory."

```
ThisWorkbook.CustomViews("Current Inventory").Show
```



# DataLabel Object

Multiple objects  [DataLabel](#)  
 Multiple objects

Represents the data label on a chart point or trendline. On a series, the **DataLabel** object is a member of the [DataLabels](#) collection. The **DataLabels** collection contains a **DataLabel** object for each point. For a series without definable points (such as an area series), the **DataLabels** collection contains a single **DataLabel** object.

## Using the DataLabel Object

Use **DataLabels**(*index*), where *index* is the data-label index number, to return a single **DataLabel** object. The following example sets the number format for the fifth data label in series one in embedded chart one on worksheet one.

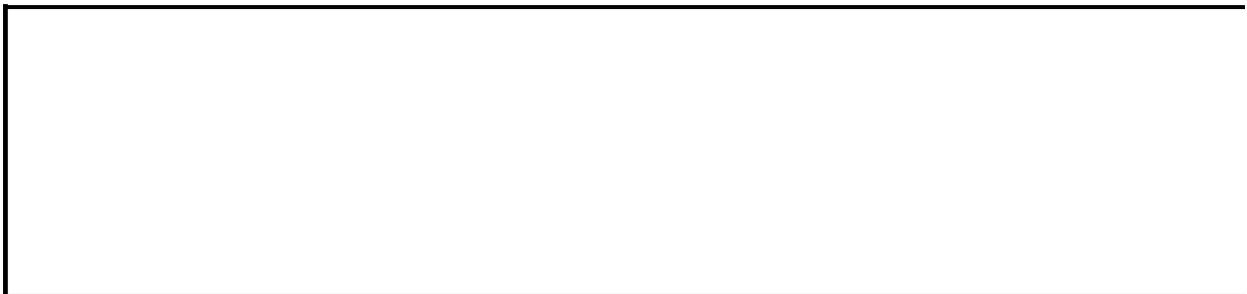
```
Worksheets(1).ChartObjects(1).Chart _  
    .SeriesCollection(1).DataLabels(5).NumberFormat = "0.000"
```

Use the **DataLabel** property to return the **DataLabel** object for a single point. The following example turns on the data label for the second point in series one on the chart sheet named "Chart1" and sets the data label text to "Saturday."

```
With Charts("chart1")  
    With .SeriesCollection(1).Points(2)  
        .HasDataLabel = True  
        .DataLabel.Text = "Saturday"  
    End With  
End With
```

On a trendline, the **DataLabel** property returns the text shown with the trendline. This can be the equation, the R-squared value, or both (if both are showing). The following example sets the trendline text to show only the equation and then places the data label text in cell A1 on the worksheet named "Sheet1."

```
With Charts("chart1").SeriesCollection(1).Trendlines(1)  
    .DisplayRSquared = False  
    .DisplayEquation = True  
    Worksheets("sheet1").Range("a1").Value = .DataLabel.Text  
End With
```



# DataTable Object

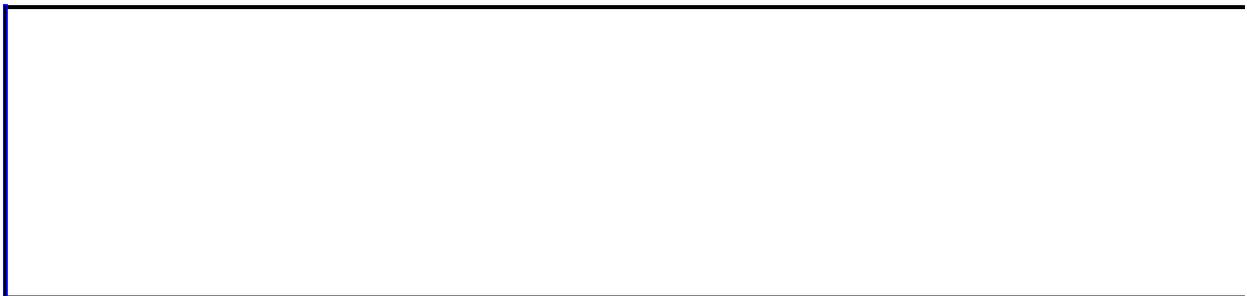
[Chart](#) └ [DataTable](#)  
└ Multiple objects

Represents a chart data table.

## Using the DataTable Object

Use the [DataTable](#) property to return a **DataTable** object. The following example adds a data table with an outline border to embedded chart one.

```
With Worksheets(1).ChartObjects(1).Chart  
    .HasDataTable = True  
    .DataTable.HasBorderOutline = True  
End With
```



# DefaultWebOptions Object

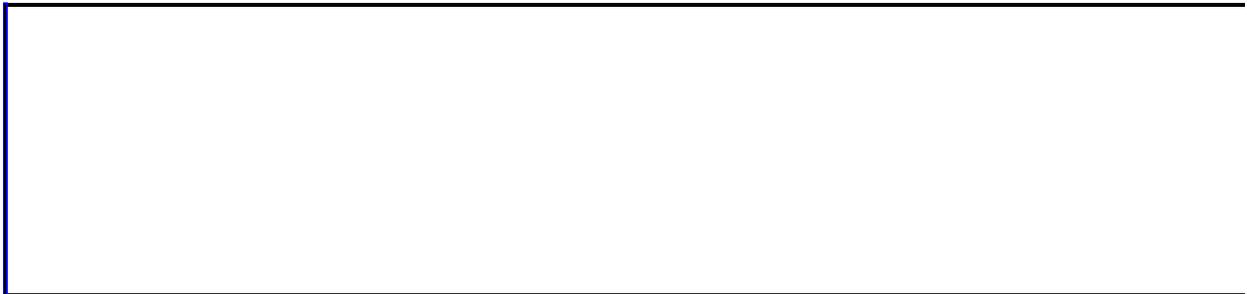
[Application](#)  [DefaultWebOptions](#)

Contains global application-level attributes used by Microsoft Excel when you save a document as a Web page or open a Web page. You can return or set attributes either at the application (global) level or at the workbook level. (Note that attribute values can be different from one workbook to another, depending on the attribute value at the time the workbook was saved.) Workbook-level attribute settings override application-level attribute settings. Workbook-level attributes are contained in the [WebOptions](#) object.

## Using the DefaultWebOptions Object

Use the [DefaultWebOptions](#) property to return the **DefaultWebOptions** object. The following example checks to see whether PNG (Portable Network Graphics) is allowed as an image format and sets the strImageFileType variable accordingly.

```
Set objAppWebOptions = Application.DefaultWebOptions
With objAppWebOptions
    If .AllowPNG = True Then
        strImageFileType = "PNG"
    Else
        strImageFileType = "JPG"
    End If
End With
```



# Diagram Object

Multiple objects [└Diagram](#)  
[└DiagramNodes](#)

Represents a diagram.

## Using the Diagram object

Use the **Diagram** property of the **Shape** object or **ShapeRange** collection to return a **Diagram** object. The following example adds a radial diagram to the active worksheet.

```
Sub NewDiagram()  
  
    Dim wksActiveSheet As Worksheet  
    Dim shDiagram As Shape  
  
    Set wksActiveSheet = ActiveSheet  
    Set shDiagram = wksActiveSheet.Shapes.AddDiagram( _  
        Type:=msoDiagramRadial, _  
        Left:=20, Top:=40, _  
        Width:=400, Height:=200)  
  
    ' Fill the diagram to make it visible to the user  
    shDiagram.Fill.Visible = msoTrue  
  
End Sub
```

You can also convert the current diagram to a different diagram by using the **Convert** method.

**Note** If the current diagram is an organization chart (**msoDiagramOrgChart**) a run-time error will occur. In this example, a radial diagram is converted into a target diagram.

```
Sub NewDiagram()  
  
    Dim wksActiveSheet As Worksheet  
    Dim shDiagram As Shape  
  
    Set wksActiveSheet = ActiveSheet  
    Set shDiagram = wksActiveSheet.Shapes.AddDiagram( _  
        Type:=msoDiagramRadial, _  
        Left:=20, Top:=40, _  
        Width:=400, Height:=200)  
  
    ' Fill the diagram to make it visible to the user  
    shDiagram.Fill.Visible = msoTrue
```

```
' Convert the diagram.  
shDiagram.Diagram.Convert Type:=msoDiagramTarget
```

End Sub

There are several types of diagrams to chose from when working with the **Diagram** object. Refer to the [AddDiagram](#) method to view a list of available diagram types.



# DiagramNode Object

Multiple objects [↳ DiagramNode](#)  
↳ Multiple objects

Represents a node in a diagram.

## Using the **DiagramNode** object

Use the [AddNode](#) method to add a node to a diagram or to a diagram node. This example assumes the third shape in the active worksheet is a diagram and adds a node to it.

```
Sub AddDiagramNode()  
    ActiveSheet.Shapes(3).DiagramNode.Children.AddNode  
End Sub
```

Use the [Delete](#) method to remove a node from a diagram or diagram node. This example assumes the second shape in the active worksheet is a diagram and removes the first node from it.

```
Sub DeleteDiagramNode()  
    ActiveSheet.Shapes(2).DiagramNode.Children(1).Delete  
End Sub
```

To return a **DiagramNode** object, use one of the following:

- The **DiagramNode** object's **AddNode**, **CloneNode**, **NextNode** or **PrevNode** methods, or **Root** property
- The **DiagramNodeChildren** collection's **AddNode** or **Item** methods, or **FirstChild** or **LastChild** properties
- The **DiagramNodes** collection's **Item** method
- The **Shape** object's or **ShapeRange** collection's **DiagramNode** property

A diagram node can terminate, or contain other child diagrams, child diagram nodes, or child shapes:

- To refer to a child diagram, use the **Diagram** property.
- To refer to an individual child diagram node, use the **AddNode**, **CloneNode**, **NextNode** or **PrevNode** methods, or **Root** property.
- To refer to a collection of child diagram nodes, use the **Children** property.
- To refer to a shape, use the **Shape** or **TextShape** properties.



# DiagramNodeChildren Collection

[DiagramNode](#) └ [DiagramNodeChildren](#)  
└ [DiagramNode](#)

A collection of **DiagramNode** objects that represents child nodes in a diagram.

## Using the DiagramNodeChildren collection

Use the **Children** property of the **DiagramNode** object to return a **DiagramNodeChildren** collection. To add an individual child diagram node to the collection, use the **AddNode** method. To return individual child diagram nodes in the collection, use the **FirstChild** or **LastChild** properties, or the **Item** method.

This example deletes the first child of the second node in the first diagram in the worksheet. This example assumes that the first shape in the active worksheet is a diagram with at least two nodes, one with child nodes.

```
Sub DiagramNodeChild()  
    ActiveSheet.Shapes(1).Diagram.Nodes.Item(2) _  
        .Children.FirstChild.Delete  
End Sub
```



# Dialog Object

[Application](#) └ [Dialogs](#)  
└ [Dialog](#)

Represents a built-in Microsoft Excel dialog box. The **Dialog** object is a member of the [Dialogs](#) collection. The **Dialogs** collection contains all the built-in dialog boxes in Microsoft Excel. You cannot create a new built-in dialog box or add one to the collection. The only useful thing you can do with a **Dialog** object is use it with the [Show](#) method to display the corresponding dialog box.

## Using the Dialog Object

Use **Dialogs**(*index*), where *index* is a built-in constant identifying the dialog box, to return a single **Dialog** object. The following example runs the built-in **Open** dialog box (**File** menu). The **Show** method returns **True** if Microsoft Excel successfully opens a file; it returns **False** if the user cancels the dialog box.

```
dlgAnswer = Application.Dialogs(xlDialogOpen).Show
```

The Microsoft Excel Visual Basic object library includes built-in constants for many of the built-in dialog boxes. Each constant is formed from the prefix "xlDialog" followed by the name of the dialog box. For example, the **Apply Names** dialog box constant is **xlDialogApplyNames**, and the **Find File** dialog box constant is **xlDialogFindFile**. These constants are members of the **XlBuiltinDialog** enumerated type. For more information about the available constants, see [Built-in Dialog Box Argument Lists](#).



# DisplayUnitLabel Object

[Axis](#) └ [DisplayUnitLabel](#)

└ Multiple objects

Represents a unit label on an axis in the specified chart. Unit labels are useful for charting large values— for example, in the millions or billions. You can make the chart more readable by using a single unit label instead of large numbers at each tick mark.

## Using the DisplayUnitLabel Object

Use the [DisplayUnitLabel](#) property to return the **DisplayUnitLabel** object. The following example sets the display label caption to "Millions" on the value axis in Chart1, and then it turns off automatic font scaling.

```
With Charts("Chart1").Axes(xlValue).DisplayUnitLabel
    .Caption = "Millions"
    .AutoScaleFont = False
End With
```



# DownBars Object

[ChartGroup](#) └ [DownBars](#)

└ Multiple objects

Represents the down bars in a chart group. Down bars connect points on the first series in the chart group with lower values on the last series (the lines go down from the first series). Only 2-D line groups that contain at least two series can have down bars. This object isn't a collection. There's no object that represents a single down bar; you either have up bars and down bars turned on for all points in a chart group or you have them turned off.

## Using the DownBars Object

Use the **DownBars** property to return the **DownBars** object. The following example turns on up and down bars for chart group one in embedded chart one on the worksheet named "Sheet5." The example then sets the up bar color to blue and the down bar color to red.

```
With Worksheets("sheet5").ChartObjects(1).Chart.ChartGroups(1)
    .HasUpDownBars = True
    .UpBars.Interior.Color = RGB(0, 0, 255)
    .DownBars.Interior.Color = RGB(255, 0, 0)
End With
```

## Remarks

If the [HasUpDownBars](#) property is **False**, most properties of the **DownBars** object are disabled.

---

---

---

# DropLines Object

[ChartGroup](#) └ [DropLines](#)  
└ [Border](#)

Represents the drop lines in a chart group. Drop lines connect the points in the chart with the x-axis. Only line and area chart groups can have drop lines. This object isn't a collection. There's no object that represents a single drop line; you either have drop lines turned on for all points in a chart group or you have them turned off.

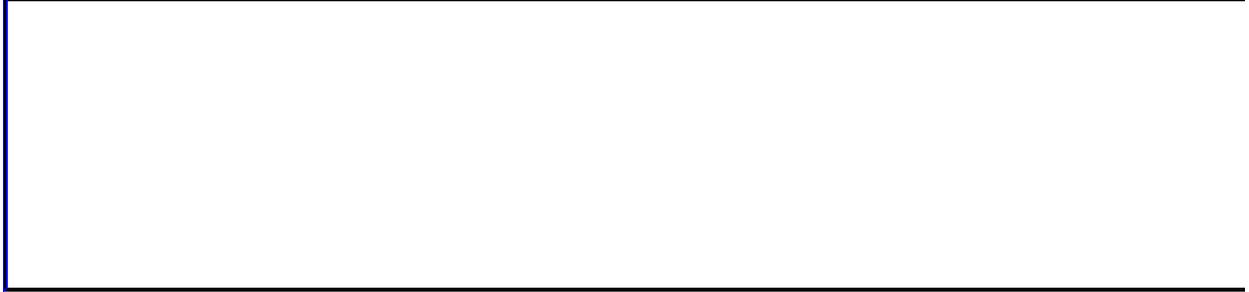
## Using the DropLines Object

Use the **DropLines** property to return the **DropLines** object. The following example turns on drop lines for chart group one in embedded chart one and then sets the drop line color to red.

```
Worksheets("sheet1").ChartObjects(1).Activate  
ActiveChart.ChartGroups(1).HasDropLines = True  
ActiveChart.ChartGroups(1).DropLines.Border.ColorIndex = 3
```

## Remarks

If the [HasDropLines](#) property is **False**, most properties of the **DropLines** object are disabled.



# Error Object

[Range](#) | [Errors](#)  
| [Error](#)

Represents a spreadsheet error for a range.

## Using the Error object

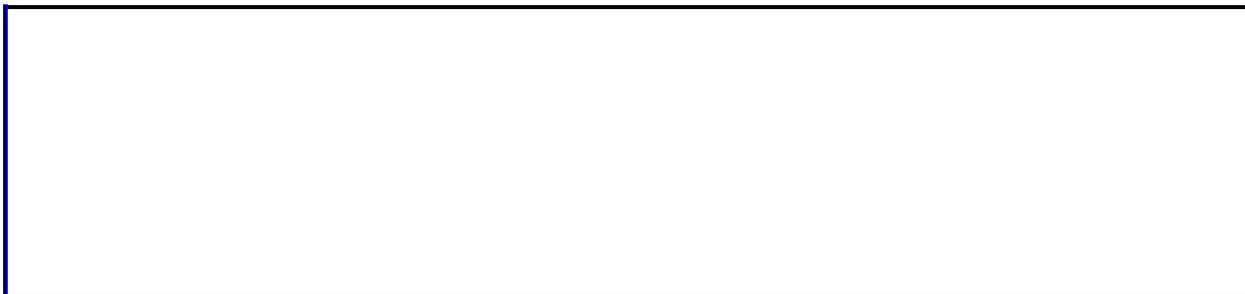
Use the [Item](#) property of the **Errors** object to return an **Error** object.

Once an **Error** object is returned, you can use the **Value** property, in conjunction with the **Errors** property to check whether a particular error checking option is enabled.

The following example creates a formula in cell A1 referencing empty cells, and then it uses **Item(index)**, where *index* identifies the error type, to display a message stating the situation.

```
Sub CheckEmptyCells()  
  
    Dim rngFormula As Range  
    Set rngFormula = Application.Range("A1")  
  
    ' Place a formula referencing empty cells.  
    Range("A1").Formula = "=A2+A3"  
    Application.ErrorCheckingOptions.EmptyCellReferences = True  
  
    ' Perform check to see if EmptyCellReferences check is on.  
    If rngFormula.Errors.Item(xlEmptyCellReferences).Value = True Then  
        MsgBox "The empty cell references error checking feature is  
    Else  
        MsgBox "The empty cell references error checking feature is  
    End If  
  
End Sub
```

**Note** Be careful not to confuse the **Error** object with error handling features of Visual Basic.



# ErrorBars Object

[Series](#) └ [ErrorBars](#)  
└ [Border](#)

Represents the error bars on a chart series. Error bars indicate the degree of uncertainty for chart data. Only series in area, bar, column, line, and scatter groups on a 2-D chart can have error bars. Only series in scatter groups can have x and y error bars. This object isn't a collection. There's no object that represents a single error bar; you either have x error bars or y error bars turned on for all points in a series or you have them turned off.

## Using the ErrorBars Object

Use the **ErrorBars** property to return the **ErrorBars** object. The following example turns on error bars for series one in embedded chart one and then sets the end style for the error bars.

```
Worksheets("sheet1").ChartObjects(1).Activate  
ActiveChart.SeriesCollection(1).HasErrorBars = True  
ActiveChart.SeriesCollection(1).ErrorBars.EndStyle = xlNoCap
```

## Remarks

The [ErrorBar](#) method changes the error bar format and type.

---

# ErrorCheckingOptions Object

[Application](#) <sup>|</sup> [ErrorCheckingOptions](#)

Represents the error-checking options for an application.

## Using the ErrorCheckingOptions Object

Use the **ErrorCheckingOptions** property of the **Application** object to return an **ErrorCheckingOptions** object.

Reference the **Item** property of the **Errors** object to view a list of index values associated with error-checking options.

Once an **ErrorCheckingOptions** object is returned, you can use the following properties, which are members of the **ErrorCheckingOptions** object, to set or return error checking options.

- [BackgroundChecking](#)
- [EmptyCellReferences](#)
- [EvaluateToError](#)
- [InconsistentFormula](#)
- [IndicatorColorIndex](#)
- [NumberAsText](#)
- [OmittedCells](#)
- [TextDate](#)
- [UnlockedFormulaCells](#)

The following example uses the [TextDate](#) property to enable error checking for two-digit-year text dates and notifies the user.

```
Sub CheckTextDates()  
  
    Dim rngFormula As Range  
    Set rngFormula = Application.Range("A1")  
  
    Range("A1").Formula = "'April 23, 00"  
    Application.ErrorCheckingOptions.TextDate = True  
  
    ' Perform check to see if 2 digit year TextDate check is on.  
    If rngFormula.Errors.Item(xlTextDate).Value = True Then  
        MsgBox "The text date error checking feature is enabled."  
    Else  
        MsgBox "The text date error checking feature is not on."  
    End If  
  
End Sub
```



# FillFormat Object

Multiple objects [FillFormat](#)  
[ColorFormat](#)

Represents fill formatting for a shape. A shape can have a solid, gradient, texture, pattern, picture, or semi-transparent fill.

## Using the FillFormat Object

Use the **Fill** property to return a **FillFormat** object. The following example adds a rectangle to myDocument and then sets the gradient and color for the rectangle's fill.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddShape(msoShapeRectangle, _
    90, 90, 90, 80).Fill
    .ForeColor.RGB = RGB(0, 128, 128)
    .OneColorGradient msoGradientHorizontal, 1, 1
End With
```

## Remarks

Many of the properties of the **FillFormat** object are read-only. To set one of these properties, you have to apply the corresponding method.



# Filter Object

[AutoFilter](#) └ [Filters](#)  
└ [Filter](#)

Represents a filter for a single column. The **Filter** object is a member of the [Filters](#) collection. The **Filters** collection contains all the filters in an autofiltered range.

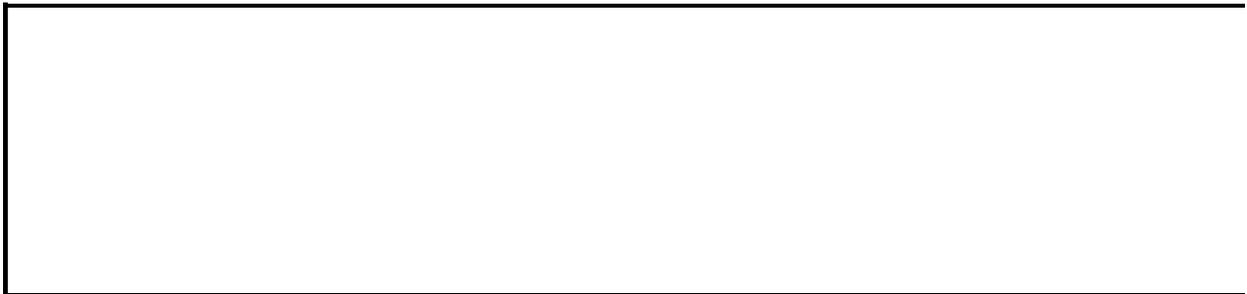
## Using the Filter Object

Use **Filters**(*index*), where *index* is the filter title or index number, to return a single **Filter** object. The following example sets a variable to the value of the **On** property of the filter for the first column in the filtered range on the Crew worksheet.

```
Set w = Worksheets("Crew")
If w.AutoFilterMode Then
    filterIsOn = w.AutoFilter.Filters(1).On
End If
```

Note that all the properties of the **Filter** object are read-only. To set these properties, apply autofiltering manually or using the **AutoFilter** method of the **Range** object, as shown in the following example.

```
Set w = Worksheets("Crew")
w.Cells.AutoFilter field:=2, Criteria1:="Crucial", _
    Operator:=xlOr, Criteria2:="Important"
```



# Floor Object

[Chart](#) └ [Floor](#)  
└ Multiple objects

Represents the floor of a 3-D chart

## Using the Floor Object

Use the **Floor** property to return the **Floor** object. The following example sets the floor color for embedded chart one to cyan. The example will fail if the chart isn't a 3-D chart.

```
Worksheets("sheet1").ChartObjects(1).Activate  
ActiveChart.Floor.Interior.Color = RGB(0, 255, 255)
```



# Font Object

Multiple objects [Font](#)

Contains the font attributes (font name, font size, color, and so on) for an object.

## Using the Font Object

Use the **Font** property to return the **Font** object. The following example formats cells A1:C5 as bold.

```
Worksheets("Sheet1").Range("A1:C5").Font.Bold = True
```

If you don't want to format all the text in a cell or graphic the same way, use the [Characters](#) property to return a subset of the text.



# FormatCondition Object

[FormatConditions](#)  $\perp$  [FormatCondition](#)

$\perp$  Multiple objects

Represents a conditional format. The **FormatCondition** object is a member of the [FormatConditions](#) collection. The **FormatConditions** collection can contain up to three conditional formats for a given range.

## Using the FormatCondition Object

Use **FormatConditions**(*index*), where *index* is the index number of the conditional format, to return a **FormatCondition** object. The following example sets format properties for an existing conditional format for cells E1:E10.

```
With Worksheets(1).Range("e1:e10").FormatConditions(1)
    With .Borders
        .LineStyle = xlContinuous
        .Weight = xlThin
        .ColorIndex = 6
    End With
    With .Font
        .Bold = True
        .ColorIndex = 3
    End With
End With
```

## Remarks

Use the [Add](#) method to create a new conditional format. If you try to create more than three conditional formats for a single range, the **Add** method fails. If a range has three formats, you can use the [Modify](#) method to change one of the formats, or you can use the **Delete** method to delete a format and then use the **Add** method to create a new format.

Use the **Font**, **Border**, and **Interior** properties of the **FormatCondition** object to control the appearance of formatted cells. Some properties of these objects aren't supported by the conditional format object model. The properties that can be used with conditional formatting are listed in the following table.

<b>Object</b>	<b>Properties</b>
<b>Font</b>	<b>Bold</b>
	<b>Color</b>
	<b>ColorIndex</b>
	<b>FontStyle</b>
	<b>Italic</b>
	<b>Strikethrough</b>
	<b>Underline</b>
	The accounting underline styles cannot be used.
	<b>Bottom</b>
	<b>Color</b>
<b>Left</b>	
<b>Right</b>	

## Style

**Border** The following border styles can be used (all others aren't supported):  
**xlNone**, **xlSolid**, **xlDash**, **xlDot**, **xlDashDot**, **xlDashDotDot**,  
**xlGray50**, **xlGray75**, and **xlGray25**.

## Top

## Weight

The following border weights can be used (all others aren't supported):  
**xlWeightHairline** and **xlWeightThin**.

## Color

## ColorIndex

**Interior** **Pattern**

## PatternColorIndex



# FreeformBuilder Object

[FreeformBuilder](#)

Represents the geometry of a freeform while it's being built.

## Using the FreeformBuilder Object

Use the [BuildFreeform](#) method to return a **FreeformBuilder** object. Use the [AddNodes](#) method to add nodes to the freeform. Use the [ConvertToShape](#) method to create the shape defined in the **FreeformBuilder** object and add it to the **Shapes** collection. The following example adds a freeform with four segments to myDocument.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.BuildFreeform(msoEditingCorner, 360, 200)
    .AddNodes msoSegmentCurve, msoEditingCorner, _
        380, 230, 400, 250, 450, 300
    .AddNodes msoSegmentCurve, msoEditingAuto, 480, 200
    .AddNodes msoSegmentLine, msoEditingAuto, 480, 400
    .AddNodes msoSegmentLine, msoEditingAuto, 360, 200
    .ConvertToShape
End With
```



# Graphic Object

[PageSetup](#) └ [Graphic](#)

Contains properties that apply to header and footer picture objects.

## Using the Graphic object

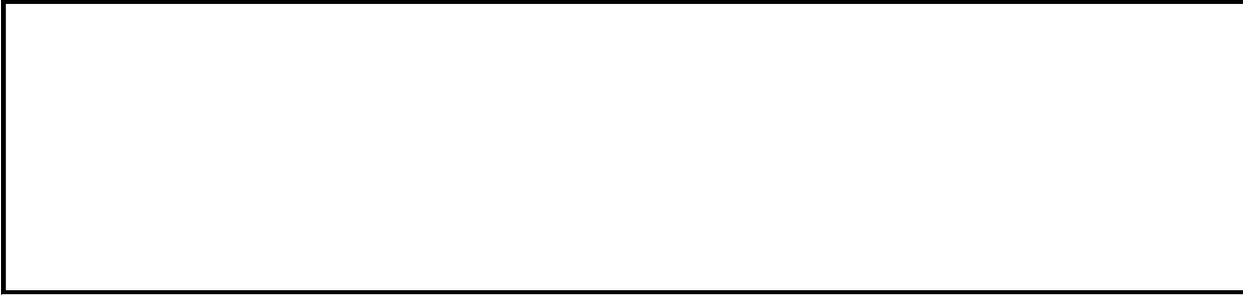
There are several properties of the **PageSetup** object that return the **Graphic** object.

Use the [CenterFooterPicture](#), [CenterHeaderPicture](#), [LeftFooterPicture](#), [LeftHeaderPicture](#), [RightFooterPicture](#), or [RightHeaderPicture](#) properties to return a **Graphic** object.

The following example adds a picture titled: Sample.jpg from the C:\ drive to the left section of the footer. This example assumes that a file called Sample.jpg exists on the C:\ drive.

```
Sub InsertPicture()  
  
    With ActiveSheet.PageSetup.LeftFooterPicture  
        .FileName = "C:\Sample.jpg"  
        .Height = 275.25  
        .Width = 463.5  
        .Brightness = 0.36  
        .ColorType = msoPictureGrayscale  
        .Contrast = 0.39  
        .CropBottom = -14.4  
        .CropLeft = -28.8  
        .CropRight = -14.4  
        .CropTop = 21.6  
    End With  
  
    ' Enable the image to show up in the left footer.  
    ActiveSheet.PageSetup.LeftFooter = "&G"  
  
End Sub
```

**Note** It is required that "&G" is a part of the **LeftFooter** string in order for the image to show up in the left footer.



# Gridlines Object

[Axis](#) └ [Gridlines](#)  
└ [Border](#)

Represents major or minor gridlines on a chart axis. Gridlines extend the tick marks on a chart axis to make it easier to see the values associated with the data markers. This object isn't a collection. There's no object that represents a single gridline; you either have all gridlines for an axis turned on or all of them turned off.

## Using the Gridlines Object

Use the **MajorGridlines** property to return the **GridLines** object that represents the major gridlines for the axis. Use the **MinorGridlines** property to return the **GridLines** object that represents the minor gridlines. It's possible to return both major and minor gridlines at the same time.

The following example turns on major gridlines for the category axis on the chart sheet named "Chart1" and then formats the gridlines to be blue dashed lines.

```
With Charts("chart1").Axes(xlCategory)
    .HasMajorGridlines = True
    .MajorGridlines.Border.Color = RGB(0, 0, 255)
    .MajorGridlines.Border.LineStyle = xlDash
End With
```



# GroupShapes Collection Object

Multiple objects [↳ GroupShapes](#)  
[↳ ShapeRange](#)

Represents the individual shapes within a grouped shape. Each shape is represented by a [Shape](#) object. Using the [Item](#) method with this object, you can work with single shapes within a group without having to ungroup them.

## Using The GroupShapes Collection

Use the [GroupItems](#) property to return the **GroupShapes** collection. Use **GroupItems(index)**, where *index* is the number of the individual shape within the grouped shape, to return a single shape from the the **GroupShapes** collection. The following example adds three triangles to myDocument, groups them, sets a color for the entire group, and then changes the color for the second triangle only.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes
    .AddShape(msoShapeIsoscelesTriangle, _
        10, 10, 100, 100).Name = "shpOne"
    .AddShape(msoShapeIsoscelesTriangle, _
        150, 10, 100, 100).Name = "shpTwo"
    .AddShape(msoShapeIsoscelesTriangle, _
        300, 10, 100, 100).Name = "shpThree"
With .Range(Array("shpOne", "shpTwo", "shpThree")).Group
    .Fill.PresetTextured msoTextureBlueTissuePaper
    .GroupItems(2).Fill.PresetTextured msoTextureGreenMarble
End With
End With
```



# HiLoLines Object

[ChartGroup](#) └ [HiLoLines](#)  
└ [Border](#)

Represents the high-low lines in a chart group. High-low lines connect the highest point with the lowest point in every category in the chart group. Only 2-D line groups can have high-low lines. This object isn't a collection. There's no object that represents a single high-low line; you either have high-low lines turned on for all points in a chart group or you have them turned off.

## Using the HiLoLines Object

Use the **HiLoLines** property to return the **HiLoLines** object. The following example uses the **AutoFormat** method to create a high-low-close stock chart in embedded chart one (the chart must contain three series) on worksheet one. The example then makes the high-low lines blue.

```
Worksheets(1).ChartObjects(1).Activate  
ActiveChart.AutoFormat gallery:=xlLine, format:=8  
ActiveChart.ChartGroups(1).HiLoLines.Border.Color = RGB(0, 0, 255)
```

## Remarks

If the [HasHiLoLines](#) property is **False**, most properties of the **HiLoLines** object are disabled.



# HPageBreak Object

Multiple objects [↳ HPageBreaks](#)

[↳ HPageBreak](#)

[↳ Multiple objects](#)

Represents a horizontal page break. The **HPageBreak** object is a member of the [HPageBreaks](#) collection.

## Using the HPageBreak Object

Use **HPageBreaks**(*index*), where *index* is the index number of the page break, to return an **HPageBreak** object. The following example changes the location of horizontal page break one.

```
Worksheets(1).HPageBreaks(1).Location = Worksheets(1).Range("e5")
```

**Note** There is a limit of 1026 horizontal page breaks per sheet.



# Hyperlink Object

Multiple objects [Hyperlink](#)  
└─ Multiple objects

Represents a hyperlink. The **Hyperlink** object is a member of the [Hyperlinks](#) collection.

## Using the Hyperlink Object

Use the [Hyperlink](#) property to return the hyperlink for a shape (a shape can have only one hyperlink). The following example activates the hyperlink for shape one.

```
Worksheets(1).Shapes(1).Hyperlink.Follow NewWindow:=True
```

A range or worksheet can have more than one hyperlink. Use **Hyperlinks**(*index*), where *index* is the hyperlink number, to return a single **Hyperlink** object. The following example activates hyperlink two in the range A1:B2.

```
Worksheets(1).Range("A1:B2").Hyperlinks(2).Follow
```



# Interior Object

Multiple objects <sup>L</sup>[Interior](#)

Represents the interior of an object.

## Using the Interior Object

Use the **Interior** property to return the **Interior** object. The following example sets the color for the interior of cell A1 to red.

```
Worksheets("Sheet1").Range("A1").Interior.ColorIndex = 3
```



# IRtdServer Object

[IRtdServer](#)

Represents an interface for a real-time data server.

## Using the IRtdServer object

The **IRTDServer** object can be instantiated or created only by implementing the **IRTDServer** interface using the **Implements** keyword.



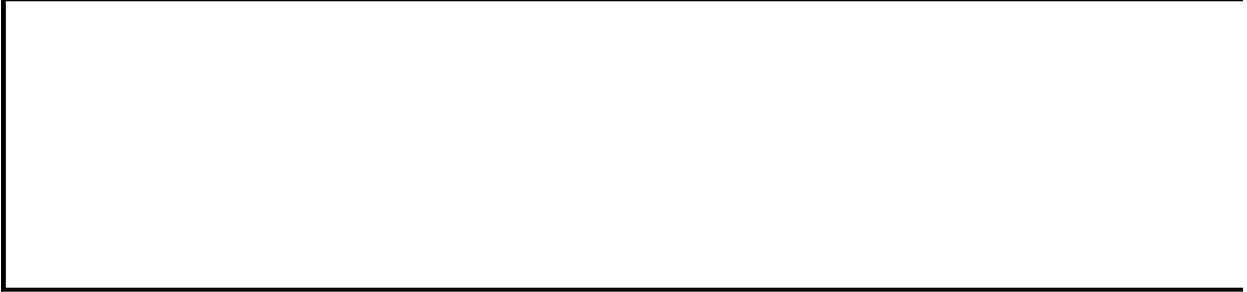
# IRTDUpdateEvent Object

[IRTDUpdateEvent](#)

Represents real-time data update events.

## Using the **IRTDUpdateEvent** object

To instantiate or to return an **IRTDUpdateEvent** object, declare a variable as an **IRTDUpdateEvent** object, and then use that variable as a callback object.



# LeaderLines Object

[Series](#) └ [LeaderLines](#)  
└ [Border](#)

Represents leader lines on a chart. Leader lines connect data labels to data points. This object isn't a collection; there's no object that represents a single leader line.

## Using the LeaderLines Object

Use the [LeaderLines](#) property to return the **LeaderLines** object. The following example adds data labels and blue leader lines to series one on chart one.

```
With Worksheets(1).ChartObjects(1).Chart.SeriesCollection(1)
    .HasDataLabels = True
    .DataLabels.Position = xlLabelPositionBestFit
    .HasLeaderLines = True
    .LeaderLines.Border.ColorIndex = 5
End With
```



# Legend Object

[Chart](#) └ [Legend](#)  
└ Multiple objects

Represents the legend in a chart. Each chart can have only one legend. The **Legend** object contains one or more [LegendEntry](#) objects; each [LegendEntry](#) object contains a [LegendKey](#) object.

## Using the Legend Object

Use the **Legend** property to return the **Legend** object. The following example sets the font style for the legend in embedded chart one on worksheet one to bold.

```
Worksheets(1).ChartObjects(1).Chart.Legend.Font.Bold = True
```

## Remarks

The chart legend isn't visible unless the [HasLegend](#) property is **True**. If this property is **False**, properties and methods of the **Legend** object will fail.



# LegendEntry Object

[LegendEntries](#) | [LegendEntry](#)  
└ Multiple objects

Represents a legend entry in a chart legend. The **LegendEntry** object is a member of the [LegendEntries](#) collection. The **LegendEntries** collection contains all the **LegendEntry** objects in the legend.

Each legend entry has two parts: the text of the entry, which is the name of the series associated with the legend entry; and an entry marker, which visually links the legend entry with its associated series or trendline in the chart. Formatting properties for the entry marker and its associated series or trendline are contained in the [LegendKey](#) object.

The text of a legend entry cannot be changed. **LegendEntry** objects support font formatting, and they can be deleted. No pattern formatting is supported for legend entries. The position and size of entries is fixed.

## Using the LegendEntry Object

Use **LegendEntries**(*index*), where *index* is the legend entry index number, to return a single **LegendEntry** object. You cannot return legend entries by name.

The index number represents the position of the legend entry in the legend. LegendEntries(1) is at the top of the legend, and LegendEntries(LegendEntries.Count) is at the bottom. The following example changes the font for the text of the legend entry at the top of the legend (this is usually the legend for series one) in embedded chart one on the worksheet named "Sheet1."

```
Worksheets("sheet1").ChartObjects(1).Chart _  
    .Legend.LegendEntries(1).Font.Italic = True
```

## Remarks

There's no direct way to return the series or trendline corresponding to the legend entry.

After legend entries have been deleted, the only way to restore them is to remove and recreate the legend that contained them by setting the [HasLegend](#) property for the chart to **False** and then back to **True**.



# LegendKey Object

[LegendEntry](#) └ [LegendKey](#)  
└ Multiple objects

Represents a legend key in a chart legend. Each legend key is a graphic that visually links a legend entry with its associated series or trendline in the chart. The legend key is linked to its associated series or trendline in such a way that changing the formatting of one simultaneously changes the formatting of the other.

## Using the LegendKey Object

Use the **LegendKey** property to return the **LegendKey** object. The following example changes the marker background color for the legend entry at the top of the legend for embedded chart one on the worksheet named "Sheet1." This simultaneously changes the format of every point in the series associated with this legend entry. The associated series must support data markers.

```
Worksheets("sheet1").ChartObjects(1).Chart _  
    .Legend.LegendEntries(1).LegendKey.MarkerBackgroundColorIndex =
```



# LineFormat Object

Multiple objects [└LineFormat](#)  
[└ColorFormat](#)

Represents line and arrowhead formatting. For a line, the **LineFormat** object contains formatting information for the line itself; for a shape with a border, this object contains formatting information for the shape's border.

## Using the LineFormat Object

Use the **Line** property to return a **LineFormat** object. The following example adds a blue, dashed line to myDocument. There's a short, narrow oval at the line's starting point and a long, wide triangle at its end point.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddLine(100, 100, 200, 300).Line
    .DashStyle = msoLineDashDotDot
    .ForeColor.RGB = RGB(50, 0, 128)
    .BeginArrowheadLength = msoArrowheadShort
    .BeginArrowheadStyle = msoArrowheadOval
    .BeginArrowheadWidth = msoArrowheadNarrow
    .EndArrowheadLength = msoArrowheadLong
    .EndArrowheadStyle = msoArrowheadTriangle
    .EndArrowheadWidth = msoArrowheadWide
End With
```



# LinkFormat Object

[Shape](#) <sup>|</sup> [LinkFormat](#)

Contains linked OLE object properties.

## Using the LinkFormat Object

Use the [LinkFormat](#) property to return the **LinkFormat** object. The following example updates an OLE object in the **Shapes** collection.

```
Worksheets(1).Shapes(1).LinkFormat.Update
```

If the **Shape** object doesn't represent a linked object, the **LinkFormat** property fails.



[Show All](#)

# ListColumn Object

[ListObject](#) └ [ListColumns](#)  
└ [ListColumn](#)  
└ Multiple objects

Represents a column in a [list](#). The **ListColumn** object is a member of the **ListColumns** collection. The **ListColumns** collection contains all the columns in a list ([ListObject](#) object).

## Using the ListColumn Object

Use the [ListColumns](#) property of the **ListObject** object to return a **ListColumns** collection.

## Example

The following example adds a new **ListColumn** object to the default **ListObject** object in the first worksheet of the active workbook. Because no position is specified, a new rightmost column is added.

```
Sub AddListColumn()  
    Dim wrksht As Worksheet  
    Dim objListCol As ListColumn  
  
    Set wrksht = ActiveWorkbook.Worksheets("Sheet1")  
    Set objListCol = wrksht.ListObjects(1).ListColumns.Add  
End Sub
```



# ListDataFormat Object

[ListColumn](#)  $\perp$  [ListDataFormat](#)

The **ListDataFormat** object holds all the data type properties of the **ListColumn** object. These properties are read-only.

## Using the ListDataFormat Object

Use the [ListDataFormat](#) property of the [ListColumn](#) object to return a **ListDataFormat** object collection. The default property of the ListDataFormat object is the **Type** property which indicates the data type of the list column. This allows the user to write code without specifying the **Type** property.

The following code example creates a linked list from a SharePoint list. It then checks to see if field 2 is required (field 1 is the ID field, which is read only). If it's a required text field, the same data is written in all existing records.

**Note** The following code example assumes that you will substitute a valid server name and the list guid in the variables **strServerName** and **strListGuid**. Additionally, the server name must be followed by `"/_vti_bin"` or the sample will not work.

```
Dim objListObject As ListObject
Dim objDataRange As Range
Dim strListGUID as String
Dim strServerName as String

strServerName = "http://<servername>/_vti_bin"
strListGUID = "{<listguid>}"

Set objListObject = Sheet1.ListObjects.Add(xlSrcExternal, _
    Array(strServerName, strListGUID), True, xlYes, Range("A1"))

With objListObject.ListColumns(2)
    Set objDataRange = .Range.Offset(1, 0).Resize(.Range.Rows.Count)
    If .ListDataFormat.Type = xlListDataTypeText And .ListDataFormat
        objDataRange.Value = "Hello World"
    End If
End With
```



# ListObject Object

Multiple objects [ListObject](#)  
└─Multiple objects

Represents a list object on a worksheet. The **ListObject** object is a member of the **ListObjects** collection. The **ListObjects** collection contains all the list objects on a worksheet.

## Using the ListObject Object

Use the [ListObjects](#) property of the **Worksheet** object to return a **ListObjects** collection. The following example adds a new **ListRow** object to the default **ListObject** object in the first worksheet of the active workbook.

```
Dim wrksht As Worksheet
Dim oListCol As ListRow

Set wrksht = ActiveWorkbook.Worksheets("Sheet1")
Set oListCol = wrksht.ListObjects(1).ListRows.Add
```



# ListRow Object



Represents a row in a List object. The **ListRow** object is a member of the **ListRows** collection. The **ListRows** collection contains all the rows in a list object.

## Using the ListRow Object

Use the [ListRows](#) property of the [ListObject](#) object to return a **ListRows Object** collection. The following example adds a new **ListRow** object to the default **ListObject** object in the first worksheet of the active workbook. Because no position is specified, a new row is added to the end of the list.

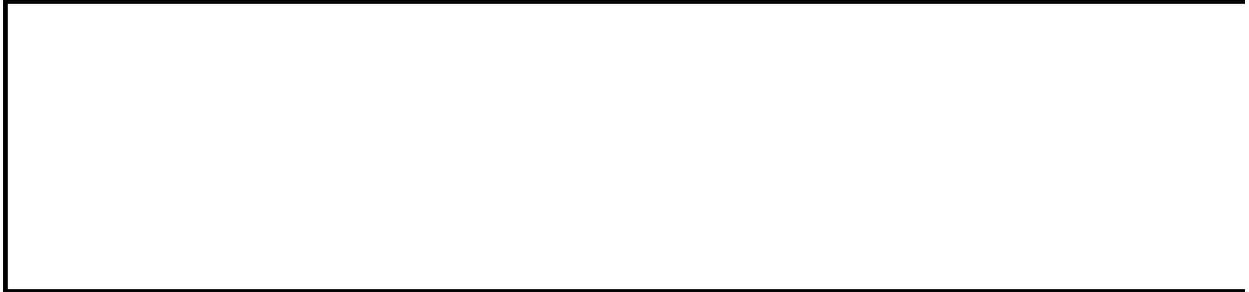
```
Dim wrksht As Worksheet
Dim oListRow As ListRow

Set wrksht = ActiveWorkbook.Worksheets("Sheet1")
Set oListRow = wrksht.ListObjects(1).ListRows.Add
```



# Mailer Object

You have requested Help for a Visual Basic keyword used only on the Macintosh. For information about this keyword, consult the language reference Help included with Microsoft Office Macintosh Edition.



# Name Object

[Names](#)  [Name](#)  
 [Range](#)

Represents a defined name for a range of cells. Names can be either built-in names— such as Database, Print\_Area, and Auto\_Open— or custom names.

## Application, Workbook, and Worksheet Objects

The **Name** object is a member of the [Names](#) collection for the **Application**, **Workbook**, and **Worksheet** objects. Use `Names(index)`, where *index* is the name index number or defined name, to return a single **Name** object.

The index number indicates the position of the name within the collection. Names are placed in alphabetic order, from a to z, and are not case-sensitive (this is the same order as is displayed in the **Define Name** and **Apply Names** dialog boxes, returned by clicking the **Name** command on the **Insert** menu). The following example displays the cell reference for the first name in the application collection.

```
MsgBox Names(1).RefersTo
```

The following example deletes the name "mySortRange" from the active workbook.

```
ActiveWorkbook.Names("mySortRange").Delete
```

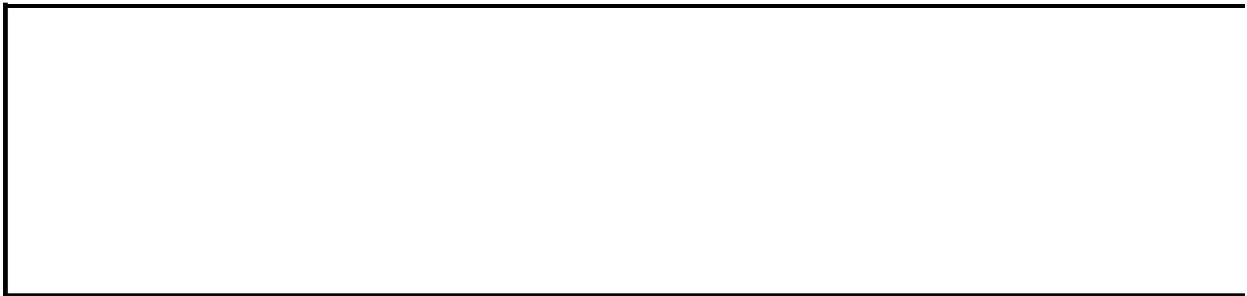
Use the **Name** property to return or set the text of the name itself. The following example changes the name of the first **Name** object in the active workbook.

```
Names(1).Name = "stock_values"
```

## Range Objects

Although a **Range** object can have more than one name, there's no **Names** collection for the **Range** object. Use **Name** with a **Range** object to return the first name from the list of names (sorted alphabetically) assigned to the range. The following example sets the **Visible** property for the first name assigned to cells A1:B1 on worksheet one.

```
Worksheets(1).Range("a1:b1").Name.Visible = False
```



# ODBCError Object

[ODBCErrors](#) | [ODBCError](#)

Represents an ODBC error generated by the most recent ODBC query. The **ODBCError** object is a member of the [ODBCErrors](#) collection. If the specified ODBC query runs without error, the **ODBCErrors** collection is empty. The errors in the collection are indexed in the order in which they're generated by the ODBC data source.

## Using the ODBCError Object

Use **ODBCErrors**(*index*), where *index* is the index number of the error, to return a single **ODBCError** object. The following example refreshes query table one and displays the first ODBC error that occurs.

```
With Worksheets(1).QueryTables(1)
    .Refresh
    If Application.ODBCErrors.Count > 0 Then
        Set er = Application.ODBCErrors(1)
        MsgBox "The following error occurred:" &
            er.ErrorString & " : " & er.SqlState
    Else
        MsgBox "Query complete: all records returned."
    End If
End With
```



# OLEDBError Object

[OLEDBErrors](#) | [OLEDBError](#)

Represents an OLE DB error returned by the most recent OLE DB query. The **OLEDBError** object is a member of the [OLEDBErrors](#) collection. If the specified OLE DB query runs without error, the **OLEDBErrors** collection is empty. The errors in the collection are indexed in the order in which they're generated by the OLE DB provider.

## Using the OLEDBError Object

Use **OLEDBErrors**(*index*), where *index* is the index number of the OLE DB error, to return a single **OLEDBError** object. The following example displays the error description and the [SqlState](#) property's value for the first error returned by the most recent OLE DB query.

```
Set objEr = Application.OLEDBErrors(1)
MsgBox "The following error occurred:" & _
    objEr.ErrorString & " : " & objEr.SqlState
```



# OLEFormat Object

[Shape](#)  [OLEFormat](#)

Contains OLE object properties.

## Using the OLEFormat Object

Use the [OLEFormat](#) property to return the **OLEFormat** object. The following example activates an OLE object in the **Shapes** collection.

```
Worksheets(1).Shapes(1).OLEFormat.Activate
```

If the **Shape** object doesn't represent a linked or embedded object, the **OLEFormat** property fails.



# OLEObject Object

[OLEObjects](#) | [OLEObject](#)  
└ Multiple objects

Represents an ActiveX control or a linked or embedded OLE object on a worksheet. The **OLEObject** object is a member of the [OLEObjects](#) collection. The **OLEObjects** collection contains all the OLE objects on a single worksheet.

## Using the OLEObject Object

Use **OLEObjects**(*index*), where *index* is the name or number of the object, to return an **OLEObject** object. The following example deletes OLE object one on Sheet1.

```
Worksheets("sheet1").OLEObjects(1).Delete
```

The following example deletes the OLE object named "ListBox1."

```
Worksheets("sheet1").OLEObjects("ListBox1").Delete
```

## Remarks

The properties and methods of the **OLEObject** object are duplicated on each ActiveX control on a worksheet. This enables Visual Basic code to gain access to these properties by using the control's name. The following example selects the check box control named "MyCheckBox," aligns it with the active cell, and then activates the control.

```
With MyCheckBox
    .Value = True
    .Top = ActiveCell.Top
    .Activate
End With
```

For more information, see [Using ActiveX controls on sheets](#).



# Outline Object

[Worksheet](#)  [Outline](#)

Represents an outline on a worksheet.

## Using the Outline Object

Use the **Outline** property to return an **Outline** object. The following example sets the outline on Sheet4 so that only the first outline level is shown.

```
Worksheets("sheet4").Outline.ShowLevels 1
```



# PageSetup Object

Multiple objects [PageSetup](#)  
[Graphic](#)

Represents the page setup description. The **PageSetup** object contains all page setup attributes (left margin, bottom margin, paper size, and so on) as properties.

## Using the PageSetup Object

Use the **PageSetup** property to return a **PageSetup** object. The following example sets the orientation to landscape mode and then prints the worksheet.

```
With Worksheets("Sheet1")
    .PageSetup.Orientation = xlLandscape
    .PrintOut
End With
```

The **With** statement makes it easier and faster to set several properties at the same time. The following example sets all the margins for worksheet one.

```
With Worksheets(1).PageSetup
    .LeftMargin = Application.InchesToPoints(0.5)
    .RightMargin = Application.InchesToPoints(0.75)
    .TopMargin = Application.InchesToPoints(1.5)
    .BottomMargin = Application.InchesToPoints(1)
    .HeaderMargin = Application.InchesToPoints(0.5)
    .FooterMargin = Application.InchesToPoints(0.5)
End With
```



# Pane Object

Multiple objects  [Pane](#)  
 [Range](#)

Represents a pane of a window. **Pane** objects exist only for worksheets and Microsoft Excel 4.0 macro sheets. The **Pane** object is a member of the [Panes](#) collection. The **Panes** collection contains all of the panes shown in a single window.

## Using the Pane Object

Use **Panes**(*index*), where *index* is the pane index number, to return a single **Pane** object. The following example splits the window in which worksheet one is displayed and then scrolls through the pane in the lower-left corner until row five is at the top of the pane.

```
Worksheets(1).Activate  
ActiveWindow.Split = True  
ActiveWindow.Panes(3).ScrollRow = 5
```



# Parameter Object

[Parameters](#) | [Parameter](#)  
| [Range](#)

Represents a single parameter used in a parameter query. The **Parameter** object is a member of the [Parameters](#) collection.

## Using the Parameter Object

Use **Parameters**(*index*), where *index* is the index number of the parameter, to return a single **Parameter** object. The following example modifies the prompt string for parameter one.

```
With Worksheets(1).QueryTables(1).Parameters(1)
    .SetParam xlPrompt, "Please " & .PromptString
End With
```



# Phonetic Object

[Range](#) └ [Phonetics](#)  
└ [Phonetic](#)  
└ [Font](#)

Contains information about a specific phonetic text string in a cell. In Microsoft Excel 97, this object contained the formatting attributes for any phonetic text in the specified range.

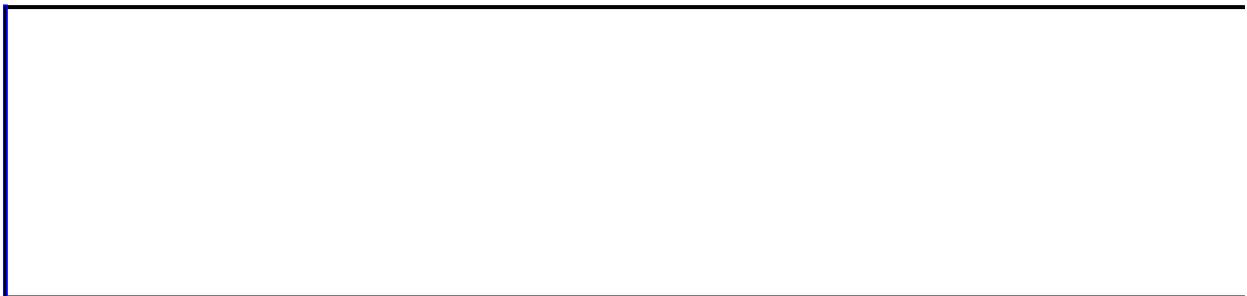
## Using the Phonetic Object

Use **Phonetics**(*index*), where *index* is the index number of the phonetic text, to return a single **Phonetic** object. The following example sets the first phonetic text string in the active cell to "フリガナ".

```
ActiveCell.Phonetics(1).Text = "フリガナ"
```

The **Phonetic** property provides compatibility with earlier versions of Microsoft Excel. You should use **Phonetics**(*index*), where *index* is the index number of the phonetic text, to return a single **Phonetic** object. To demonstrate compatibility with earlier versions of Microsoft Excel, the following example adds Furigana characters to the range A1:C4. If you add Furigana characters to a range, a new **Phonetic** object is automatically created.

```
With Range("A1:C4").Phonetic  
    .CharacterType = xlHiragana  
    .Alignment = xlPhoneticAlignCenter  
    .Font.Name = "MS Pゴシック"  
    .Font.FontStyle = "標準"  
    .Font.Size = 6  
    .Font.Strikethrough = False  
    .Font.Underline = xlUnderlineStyleNone  
    .Font.ColorIndex = xlAutomatic  
    .Visible = True  
End With
```



# PictureFormat Object

Multiple objects [PictureFormat](#)

Contains properties and methods that apply to pictures and OLE objects. The [LinkFormat](#) object contains properties and methods that apply to linked OLE objects only. The [OLEFormat](#) object contains properties and methods that apply to OLE objects whether or not they're linked.

## Using the PictureFormat Object

Use the **PictureFormat** property to return a **PictureFormat** object. The following example sets the brightness, contrast, and color transformation for shape one on myDocument and crops 18 points off the bottom of the shape. For this example to work, shape one must be either a picture or an OLE object.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(1).PictureFormat
    .Brightness = 0.3
    .Contrast = 0.7
    .ColorType = msoPictureGrayscale
    .CropBottom = 18
```



# PivotCache Object

[PivotCaches](#) | [PivotCache](#)

Represents the memory cache for a PivotTable report. The **PivotCache** object is a member of the [PivotCaches](#) collection.

## Using the PivotCache Object

Use the [PivotCache](#) method to return a **PivotCache** object for a PivotTable report (each report has only one cache). The following example causes the first PivotTable report on the first worksheet to refresh itself whenever its file is opened.

```
Worksheets(1).PivotTables(1).PivotCache.RefreshOnFileOpen = True
```

Use **PivotCaches**(*index*), where *index* is the PivotTable cache number, to return a single **PivotCache** object from the **PivotCaches** collection for a workbook. The following example refreshes cache one.

```
ActiveWorkbook.PivotCaches(1).Refresh
```



# PivotCell Object

[Range](#) └ [PivotCell](#)  
└ Multiple objects

Represents a cell in a PivotTable report.

## Using the PivotCell object

Use the [PivotCell](#) property of the [Range](#) collection to return a **PivotCell** object.

Once a **PivotCell** object is returned, you can use the [PivotCellType](#) property to determine what type of cell a particular range is. The following example determines if cell A5 in the PivotTable is a data item and notifies the user. This example assumes that a PivotTable exists on the active worksheet and that cell A5 is contained in the PivotTable. If cell A5 is not in the PivotTable, the example handles the run-time error.

```
Sub CheckPivotCellType()  
  
    On Error GoTo Not_In_PivotTable  
  
    ' Determine if cell A5 is a data item in the PivotTable.  
    If Application.Range("A5").PivotCell.PivotCellType = xlPivotCell  
        MsgBox "The PivotCell at A5 is a data item."  
    Else  
        MsgBox "The PivotCell at A5 is not a data item."  
    End If  
    Exit Sub  
  
Not_In_PivotTable:  
    MsgBox "The chosen cell is not in a PivotTable."  
  
End Sub
```

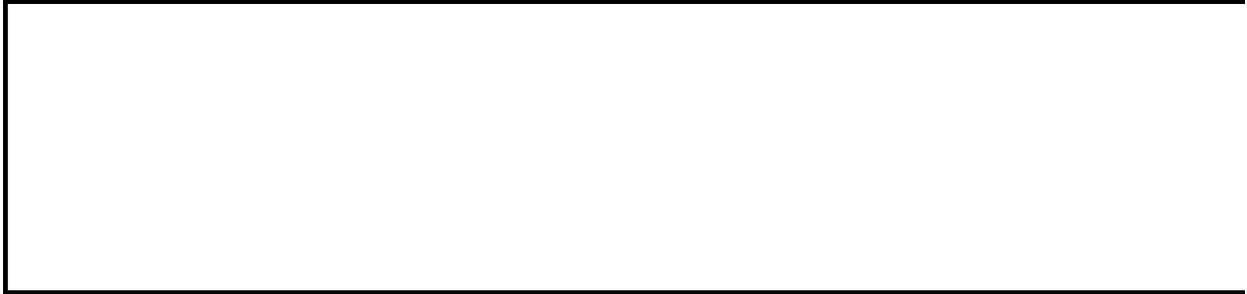
Once a **PivotCell** object is returned, you can use the [ColumnItems](#) or [RowItems](#) property to determine the [PivotItems](#) collection that corresponds to the items on the column or row axis that represents the selected number. The following example uses the [ColumnItems](#) property of the **PivotCell** object to return a [PivotItemList](#) collection.

This example determines the column field that the data item of cell B5 is in. It then determines if the column field title matches "Inventory" and notifies the user. The example assumes that a PivotTable exists on the active worksheet and that column B of the worksheet contains a column field of the PivotTable.

```
Sub CheckColumnItems()
```

```
' Determine if there is a match between the item and column field
If Application.Range("B5").PivotCell.ColumnItems.Item(1) = "Inventory"
    MsgBox "Item in B5 is a member of the 'Inventory' column field"
Else
    MsgBox "Item in B5 is not a member of the 'Inventory' column field"
End If
```

End Sub



# PivotField Object

Multiple objects [PivotField](#)

└─Multiple objects

Represents a field in a PivotTable report. The **PivotField** object is a member of the [PivotFields](#) collection. The **PivotFields** collection contains all the fields in a PivotTable report, including hidden fields.

## Using the PivotField Object

Use **PivotFields**(*index*), where *index* is the field name or index number, to return a single **PivotField** object. The following example makes the Year field a row field in the first PivotTable report on Sheet3.

```
Worksheets("sheet3").PivotTables(1) _  
    .PivotFields("year").Orientation = xlRowField
```

In some cases, it may be easier to use one of the properties that returns a subset of the PivotTable fields. The following properties are available:

- [ColumnFields](#) property
- [DataFields](#) property
- [HiddenFields](#) property
- [PageFields](#) property
- [RowFields](#) property
- [VisibleFields](#) property



[Show All](#)

# PivotFormula Object

[PivotFormulas](#) | [PivotFormula](#)

Represents a formula used to calculate results in a PivotTable report.

## Remarks

This object and its associated properties and methods aren't available for [OLAP](#) data sources because calculated fields and items aren't supported.

## Using the PivotFormula Object

Use **PivotFormulas**(*index*), where *index* is the formula number or string on the left side of the formula, to return the **PivotFormula** object. The following example changes the index number for formula one in the first PivotTable report on the first worksheet so that this formula will be solved after formula two.

```
Worksheets(1).PivotTables(1).PivotFormulas(1).Index = 2
```



# PivotItem Object

Multiple objects [PivotItem](#)  
└─ Multiple objects

Represents an item in a PivotTable field. The items are the individual data entries in a field category. The **PivotItem** object is a member of the [PivotItems](#) collection. The **PivotItems** collection contains all the items in a **PivotField** object.

## Using the PivotItem Object

Use **PivotItems**(*index*), where *index* is the item index number or name, to return a single **PivotItem** object. The following example hides all entries in the first PivotTable report on Sheet3 that contain "1998" in the Year field.

```
Worksheets("sheet3").PivotTables(1) _  
    .PivotFields("year").PivotItems("1998").Visible = False
```



# PivotLayout Object

[Chart](#) └ [PivotLayout](#)  
└ [PivotTable](#)

Represents the placement of fields in a PivotChart report.

## Using the PivotLayout Object

Use the [PivotLayout](#) property to return a **PivotLayout** object. The following example creates a list of PivotTable field names used in the first PivotChart report.

```
Sub ListFieldNames

    Dim objNewSheet As Worksheet
    Dim intRow As Integer
    Dim objPF As PivotField

    Set objNewSheet = Worksheets.Add

    intRow = 1

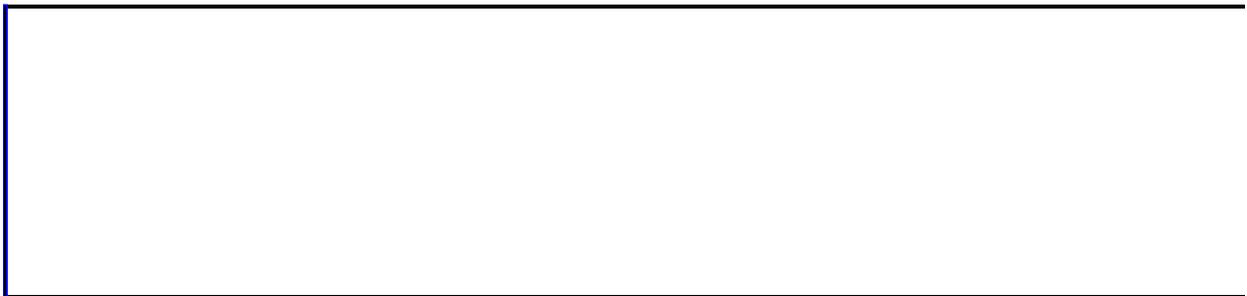
    For Each objPF In _
        Charts("Chart1").PivotLayout.PivotFields

        objNewSheet.Cells(intRow, 1).Value = objPF.Caption

        intRow = intRow + 1

    Next objPF

End Sub
```



# PivotTable Object

Multiple objects [PivotTable](#)

└─Multiple objects

Represents a PivotTable report on a worksheet. The **PivotTable** object is a member of the [PivotTables](#) collection. The **PivotTables** collection contains all the **PivotTable** objects on a single worksheet.

## Using the PivotTable Object

Use **PivotTables**(*index*), where *index* is the PivotTable index number or name, to return a single **PivotTable** object. The following example makes the field named year a row field in the first PivotTable report on Sheet3.

```
Worksheets("Sheet3").PivotTables(1) _  
    .PivotFields("Year").Orientation = xlRowField
```

## Remarks

Because PivotTable report programming can be complex, it's generally easiest to record PivotTable report actions and then revise the recorded code. To record a macro, point to **Macro** on the **Tools** menu and then click **Record New Macro**.

---

# PlotArea Object

[Chart](#) └ [PlotArea](#)  
└ Multiple objects

Represents the plot area of a chart. This is the area where your chart data is plotted. The plot area on a 2-D chart contains the data markers, gridlines, data labels, trendlines, and optional chart items placed in the chart area. The plot area on a 3-D chart contains all the above items plus the walls, floor, axes, axis titles, and tick-mark labels in the chart.

The plot area is surrounded by the chart area. The chart area on a 2-D chart contains the axes, the chart title, the axis titles, and the legend. The chart area on a 3-D chart contains the chart title and the legend. For information about formatting the chart area, see the [ChartArea](#) object.

## Using the PlotArea Object

Use the **PlotArea** property to return a **PlotArea** object. The following example activates the chart sheet named "Chart1," places a dashed border around the chart area of the active chart, and places a dotted border around the plot area.

```
Charts("Chart1").Activate  
With ActiveChart  
    .ChartArea.Border.LineStyle = xlDash  
    .PlotArea.Border.LineStyle = xlDot  
End With
```



# Point Object

[Points](#) └ [Point](#)  
└ Multiple objects

Represents a single point in a series in a chart. The **Point** object is a member of the [Points](#) collection. The **Points** collection contains all the points in one series.

## Using the Point Object

Use **Points**(*index*), where *index* is the point index number, to return a single **Point** object. Points are numbered from left to right on the series. `Points(1)` is the leftmost point, and `Points(Points.Count)` is the rightmost point. The following example sets the marker style for the third point in series one in embedded chart one on worksheet one. The specified series must be a 2-D line, scatter, or radar series.

```
Worksheets(1).ChartObjects(1).Chart. _  
    SeriesCollection(1).Points(3).MarkerStyle = xlDiamond
```



# Protection Object

[Worksheet](#) └ [Protection](#)  
└ [AllowEditRanges](#)

Represents the various types of protection options available for a worksheet.

## Using the Protection object

Use the [Protection](#) property of the [Worksheet](#) object to return a **Protection** object.

Once a **Protection** object is returned, you can use its following properties, to set or return protection options.

- [AllowDeletingColumns](#)
- [AllowDeletingRows](#)
- [AllowFiltering](#)
- [AllowFormattingCells](#)
- [AllowFormattingColumns](#)
- [AllowFormattingRows](#)
- [AllowInsertingColumns](#)
- [AllowInsertingHyperlinks](#)
- [AllowInsertingRows](#)
- [AllowSorting](#)
- [AllowUsingPivotTables](#)

The following example demonstrates how to use the [AllowInsertingColumns](#) property of the **Protection** object, placing three numbers in the top row and protecting the worksheet. Then this example checks to see if the protection setting for allowing the insertion of columns is **False** and sets it to **True**, if necessary. Finally, it notifies the user to insert a column.

```
Sub SetProtection()
```

```
    Range("A1").Formula = "1"  
    Range("B1").Formula = "3"  
    Range("C1").Formula = "4"  
    ActiveSheet.Protect
```

```
    ' Check the protection setting of the worksheet and act according  
    If ActiveSheet.Protection.AllowInsertingColumns = False Then  
        ActiveSheet.Protect AllowInsertingColumns:=True  
        MsgBox "Insert a column between 1 and 3"  
    Else  
        MsgBox "Insert a column between 1 and 3"  
    End If
```

End Sub



# PublishObject Object

[Workbook](#) └ [PublishObjects](#)  
└ [PublishObject](#)

Represents an item in a workbook that has been saved to a Web page and can be refreshed according to values specified by the properties and methods of the **PublishObject** object. The **PublishObject** object is a member of the [PublishObjects](#) collection.

## Using the PublishObject Object

Use **PublishObjects**(*index*), where *index* is the index number of the specified item in the workbook, to return a single **PublishObject** object. The following example sets the location where the first item in workbook three is saved.

```
Workbooks(3).PublishObjects(1).FileName = _  
    "\\myserver\public\finacct\statemnt.htm"
```



# QueryTable Object

Multiple objects [QueryTable](#)

└─ Multiple objects

Represents a worksheet table built from data returned from an external data source, such as an SQL server or a Microsoft Access database. The **QueryTable** object is a member of the [QueryTables](#) collection.

## Using the QueryTable Object

Use **QueryTables**(*index*), where *index* is the index number of the query table, to return a single **QueryTable** object. The following example sets query table one so that formulas to the right of it are automatically updated whenever it's refreshed.

```
Sheets("sheet1").QueryTables(1).FillAdjacentFormulas = True
```



# RecentFile Object

[Application](#) └ [RecentFiles](#)  
└ [RecentFile](#)

Represents a file in the list of recently used files. The **RecentFile** object is a member of the [RecentFiles](#) collection.

## Using the RecentFile Object

Use **RecentFiles**(*index*), where *index* is the file number, to return a **RecentFile** object. The following example opens file two in the list of recently used files.

```
Application.RecentFiles(2).Open
```



# RoutingSlip Object

[Workbook](#) | [RoutingSlip](#)

Represents the routing slip for a workbook. The routing slip is used to send a workbook through the electronic mail system.

## Using the RoutingSlip Object

Use the **RoutingSlip** property to return the **RoutingSlip** object. The following example sets the delivery style for the routing slip attached to the active workbook. For a more detailed example, see the **RoutingSlip** property.

```
ActiveWorkbook.HasRoutingSlip = True  
ActiveWorkbook.RoutingSlip.Delivery = xlOneAfterAnother
```

## Remarks

The **RoutingSlip** object doesn't exist and cannot be returned unless the [HasRoutingSlip](#) property for the workbook is **True**.

--

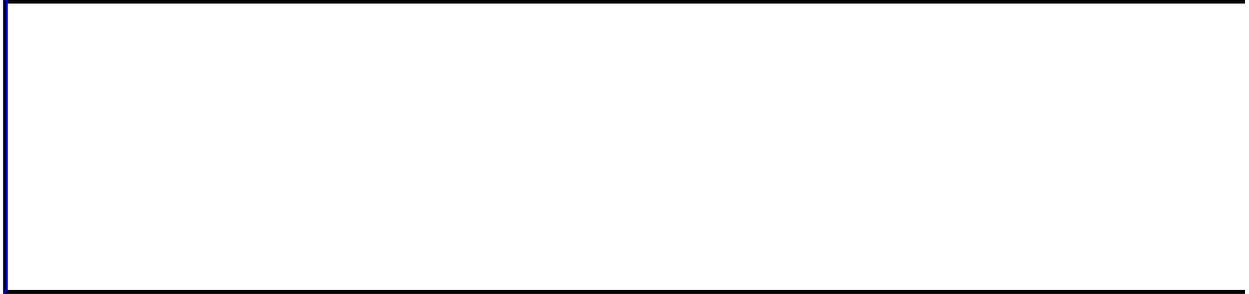
# RTD Object

[Application](#) <sup>L</sup>[RTD](#)

Represents a real-time data object.

## Using the RTD object

Use the [RTD](#) property of the [Application](#) object to return a **RTD** object.



# Scenario Object

[Scenarios](#) └ [Scenario](#)  
└ [Range](#)

Represents a scenario on a worksheet. A scenario is a group of input values (called *changing cells*) that's named and saved. The **Scenario** object is a member of the [Scenarios](#) collection. The **Scenarios** collection contains all the defined scenarios for a worksheet.

## Using the Scenario Object

Use **Scenarios**(*index*), where *index* is the scenario name or index number, to return a single **Scenario** object. The following example shows the scenario named "Typical" on the worksheet named "Options."

```
Worksheets("options").Scenarios("typical").Show
```



# Series Object

[SeriesCollection](#) └ [Series](#)  
└ Multiple objects

Represents a series in a chart. The **Series** object is a member of the [SeriesCollection](#) collection.

## Using the Series Object

Use **SeriesCollection**(*index*), where *index* is the series index number or name, to return a single **Series** object. The following example sets the color of the interior for the first series in embedded chart one on Sheet1.

```
Worksheets("sheet1").ChartObjects(1).Chart. _  
    SeriesCollection(1).Interior.Color = RGB(255, 0, 0)
```

The series index number indicates the order in which the series were added to the chart. `SeriesCollection(1)` is the first series added to the chart, and `SeriesCollection(SeriesCollection.Count)` is the last one added.



# SeriesLines Object

[ChartGroup](#) └ [SeriesLines](#)  
└ [Border](#)

Represents series lines in a chart group. Series lines connect the data values from each series. Only 2-D stacked bar or column chart groups can have series lines. This object isn't a collection. There's no object that represents a single series line; you either have series lines turned on for all points in a chart group or you have them turned off.

## Using the SeriesLines Object

Use the **SeriesLines** property to return a **SeriesLines** object. The following example adds series lines to chart group one in embedded chart one on worksheet one (the chart must be a 2-D stacked bar or column chart).

```
With Worksheets(1).ChartObjects(1).Chart.ChartGroups(1)
    .HasSeriesLines = True
    .SeriesLines.Border.Color = RGB(0, 0, 255)
End With
```

## Remarks

If the [HasSeriesLines](#) property is **False**, most properties of the **SeriesLines** object are disabled.

# ShadowFormat Object

Multiple objects [↳ShadowFormat](#)  
[↳ColorFormat](#)

Represents shadow formatting for a shape.

## Using the ShadowFormat Object

Use the **Shadow** property to return a **ShadowFormat** object. The following example adds a shadowed rectangle to myDocument. The semitransparent, blue shadow is offset 5 points to the right of the rectangle and 3 points above it.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddShape(msoShapeRectangle, _
    50, 50, 100, 200).Shadow
    .ForeColor.RGB = RGB(0, 0, 128)
    .OffsetX = 5
    .OffsetY = -3
    .Transparency = 0.5
    .Visible = True
End With
```



# Shape Object

Multiple objects [Shape](#)  
└─Multiple objects

Represents an object in the drawing layer, such as an AutoShape, freeform, OLE object, or picture. The **Shape** object is a member of the [Shapes](#) collection. The **Shapes** collection contains all the shapes on a slide.

**Note** There are three objects that represent shapes: the **Shapes** collection, which represents all the shapes on a document; the [ShapeRange](#) collection, which represents a specified subset of the shapes on a document (for example, a **ShapeRange** object could represent shapes one and four on the document, or it could represent all the selected shapes on the document); and the **Shape** object, which represents a single shape on a document. If you want to work with several shapes at the same time or with shapes within the selection, use a **ShapeRange** collection. For an overview of how to work with either a single shape or with more than one shape at a time, see [Working with Shapes \(Drawing Objects\)](#).

## Using the Shape Object

This section describes how to:

- Return an existing shape.
- Return a shape within the selection.
- Return the shapes attached to the ends of a connector.
- Return a newly created freeform.
- Return a single shape from within a group.
- Return a newly formed group of shapes.

## Returning an Existing Shape

Use **Shapes**(*index*), where *index* is the shape name or the index number, to return a **Shape** object that represents a shape. The following example horizontally flips shape one and the shape named Rectangle 1 on myDocument.

```
Set myDocument = Worksheets(1)
myDocument.Shapes(1).Flip msoFlipHorizontal
myDocument.Shapes("Rectangle 1").Flip msoFlipHorizontal
```

Each shape is assigned a default name when you add it to the **Shapes** collection. To give the shape a more meaningful name, use the **Name** property. The following example adds a rectangle to myDocument, gives it the name Red Square, and then sets its foreground color and line style.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddShape(msoShapeRectangle, _
    144, 144, 72, 72)
    .Name = "Red Square"
    .Fill.ForeColor.RGB = RGB(255, 0, 0)
    .Line.DashStyle = msoLineDashDot
End With
```

## Returning a Shape Within the Selection

Use **Selection.ShapeRange(*index*)**, where *index* is the shape name or the index number, to return a **Shape** object that represents a shape within the selection. The following example sets the fill for the first shape in the selection in the active window, assuming that there's at least one shape in the selection.

```
ActiveWindow.Selection.ShapeRange(1).Fill.ForeColor.RGB = _  
    RGB(255, 0, 0)
```

## Returning the Shapes Attached to the Ends of a Connector

To return a **Shape** object that represents one of the shapes attached by a connector, use the [BeginConnectedShape](#) or [EndConnectedShape](#) property.

## Returning a newly created freeform

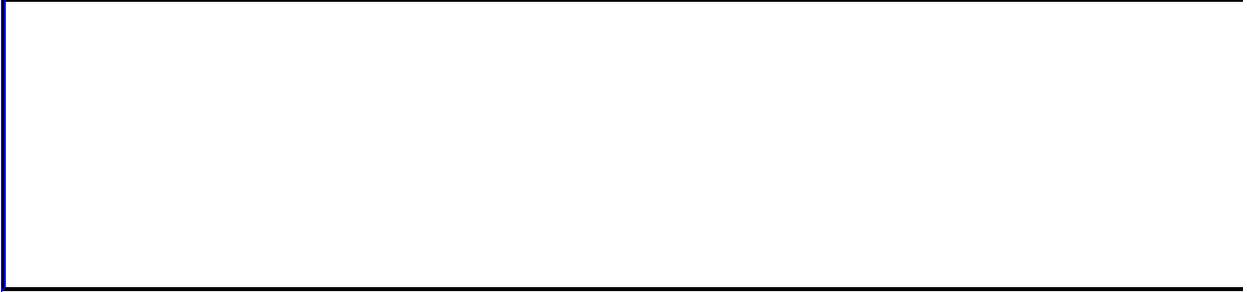
Use the [BuildFreeform](#) and [AddNodes](#) methods to define the geometry of a new freeform, and use the [ConvertToShape](#) method to create the freeform and return the **Shape** object that represents it.

## Returning a Single Shape from Within a Group

Use **GroupItems**(*index*), where *index* is the shape name or the index number within the group, to return a **Shape** object that represents a single shape in a grouped shape.

## Returning a Newly Formed Group of Shapes

Use the [Group](#) or [Regroup](#) method to group a range of shapes and return a single **Shape** object that represents the newly formed group. After a group has been formed, you can work with the group the same way you work with any other shape.



# ShapeNode Object

[ShapeNodes](#) └ [ShapeNode](#)

Represents the geometry and the geometry-editing properties of the nodes in a user-defined freeform. Nodes include the vertices between the segments of the freeform and the control points for curved segments. The **ShapeNode** object is a member of the [ShapeNodes](#) collection. The **ShapeNodes** collection contains all the nodes in a freeform.

## Using the ShapeNode Object

Use **Nodes**(*index*), where *index* is the node index number, to return a single **ShapeNode** object. If node one in shape three on myDocument is a corner point, the following example makes it a smooth point. For this example to work, shape three must be a freeform.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(3)
    If .Nodes(1).EditingType = msoEditingCorner Then
        .Nodes.SetEditingType 1, msoEditingSmooth
    End If
End With
```



# SmartTag Object

[SmartTags](#) | [SmartTag](#)  
└ Multiple objects

Represents an identifier that is assigned to a cell.

## Using the SmartTag object

Use the **Add** method of the **SmartTags** collection to return a **SmartTag** object.

Once a **SmartTag** object is returned, you can store extra metadata to a smart tag by using the **Add** method with the **Properties** property.

See the following example for a demonstration of this feature. This example adds a smart tag titled "MSFT" to cell A1, then adds extra metadata called "Market" with the value of "Nasdaq" to the smart tag and then returns the value of the property to the user. This example assumes the host system is connected to the Internet.

```
Sub UseProperties()  
  
    Dim strLink As String  
    Dim strType As String  
  
    ' Define SmartTag variables.  
    strLink = "urn:schemas-microsoft-com:smartrtags#StockTickerSymbol"  
    strType = "stockview"  
  
    ' Enable smart tags to be embedded and recognized.  
    ActiveWorkbook.SmartTagOptions.EmbedSmartTags = True  
    Application.SmartTagRecognizers.Recognize = True  
  
    Range("A1").Formula = "MSFT"  
  
    ' Add a property for MSFT smart tag and define its value.  
    Range("A1").SmartTags.Add(strLink).Properties.Add _  
        Name:="Market", Value:="Nasdaq"  
  
    ' Notify the user of the smart tag's value.  
    MsgBox Range("A1").SmartTags.Add(strLink).Properties("Market").Value  
End Sub
```

To view the extra metadata, use the **XML** property of the **SmartTag** object. This example, which builds upon the previous example, displays the extra metadata that was added to the smart tag in cell A1. The metadata for this smart tag represents the XML that would be passed to the action handler. This example assumes the host system is connected to the Internet.

```
Sub CheckXML()  
  
    Dim strLink As String  
    Dim strType As String  
  
    ' Define SmartTag variables.  
    strLink = "urn:schemas-microsoft-com:smarttags#StockTickerSymbol"  
    strType = "stockview"  
  
    ' Enable smart tags to be embedded and recognized.  
    ActiveWorkbook.SmartTagOptions.EmbedSmartTags = True  
    Application.SmartTagRecognizers.Recognize = True  
  
    Range("A1").Formula = "MSFT"  
  
    ' Display the sample of the XML.  
    MsgBox Range("A1").SmartTags.Add(strLink).XML  
  
End Sub
```



# SmartTagAction Object

[SmartTag](#) └ [SmartTagActions](#)  
└ [SmartTagAction](#)

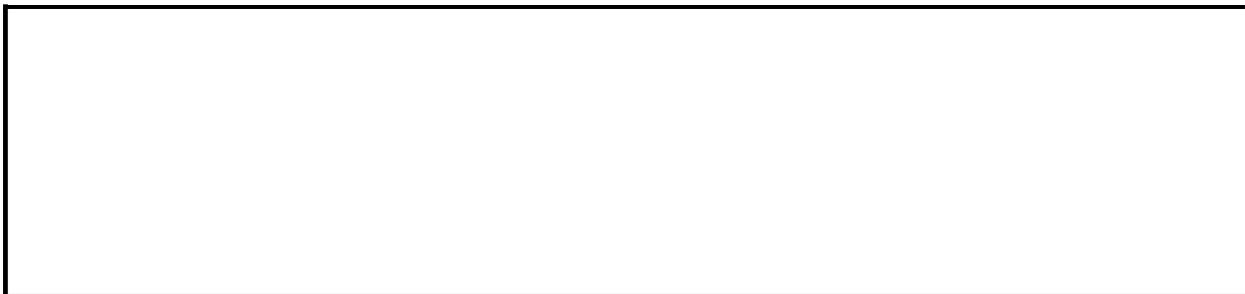
Represents the actions that can be performed with smart tags.

## Using the SmartTagAction object

Use the **Item** property of the **SmartTagActions** collection to return a **SmartTagAction** object.

Once a **SmartTagAction** object has been returned, you can activate a smart tag to automatically annotate data using the **Execute** method. This example inserts a refreshable stock quote for the ticker symbol "MSFT" and it assumes the host system is connected to the Internet.

```
Sub ExecuteASmartTag()  
  
    Dim strAction As String  
  
    strAction = "Insert refreshable stock price"  
  
    ' Enable smart tags to be embedded and recognized.  
    ActiveWorkbook.SmartTagOptions.EmbedSmartTags = True  
    Application.SmartTagRecognizers.Recognize = True  
  
    ' Invoke a smart tag for the Microsoft ticker symbol.  
    With Range("A1")  
        .Formula = "MSFT"  
        .SmartTags( _  
            "urn:schemas-microsoft-com:office:smarttags#stockticker"  
        ).SmartTagActions(strAction).Execute  
    End With  
  
End Sub
```



# SmartTagOptions Object

[Workbook](#) | [SmartTagOptions](#)

Represents the options that can be performed with smart tags.

## Using the SmartTagOptions object

Use the **SmartTagOptions** property of the **Workbook** object to return a **SmartTagOptions** object.

Once a **SmartTagOptions** object is returned, you can use the following properties to determine the display options of smart tags and whether or not to have smart tags be embedded on the active workbook.

- [EmbedSmartTags](#)
- [DisplaySmartTags](#)

This example enables the ability to embed smart tags on the active workbook and then checks the display settings for smart tags.

```
Sub CheckDisplayOptions()  
  
    'Enable SmartTags to be embedded on the active workbook.  
    ActiveWorkbook.SmartTagOptions.EmbedSmartTags = True  
  
    ' Check the display options for smart tags.  
    Select Case ActiveWorkbook.SmartTagOptions.DisplaySmartTags  
        Case xlButtonOnly  
            MsgBox "The button for smart tags will only be displayed  
        Case xlDisplayNone  
            MsgBox "Nothing will be displayed for smart tags."  
        Case xlIndicatorAndButton  
            MsgBox "The button and indicator will be displayed for s  
    End Select  
  
End Sub
```



# SmartTagRecognizer Object

[Application](#) └ [SmartTagRecognizers](#)  
└ [SmartTagRecognizer](#)

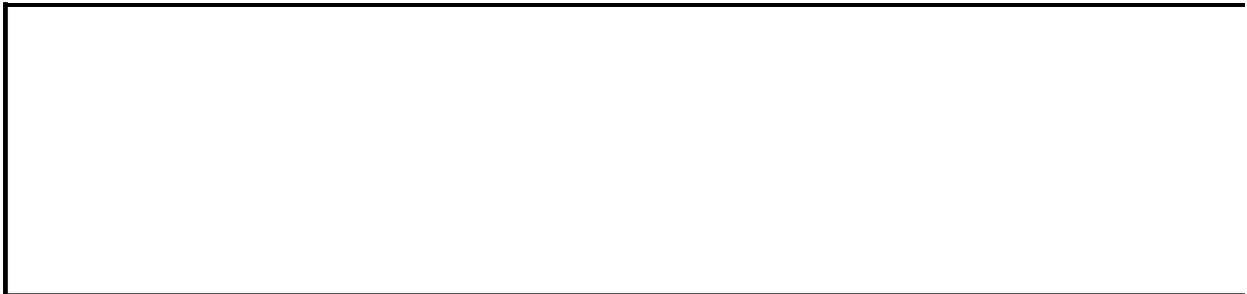
Represents recognition engines which label data with types of information as you work in Microsoft Excel.

## Using the SmartTagRecognizer object

Use the **Item**(*index*) property of the **SmartTagRecognizers** collection to return a single **SmartTagRecognizer** object.

Once a **SmartTagRecognizer** object is returned, you can determine if smart tag recognizers are enabled for the application. This example determines if smart tag recognizers are enabled and notifies the user.

```
Sub Check_SmartTagRecognizers()  
    ' Determine if smart tag recognizers are enabled.  
    If Application.SmartTagRecognizers.Item(1).Enabled = True Then  
        MsgBox "Smart tag recognizers are enabled."  
    Else  
        MsgBox "Smart tag recognizers are not enabled."  
    End If  
End Sub
```



# SoundNote Object

This object should not be used. Sound notes have been removed from Microsoft Excel.



# Speech Object

[Application](#) ↳ [Speech](#)

Contains methods and properties that pertain to speech.

## Using the Speech object

Use the **Speech** property of the **Application** object to return a **Speech** object.

Once a **Speech** object is returned, you can use the **Speak** method of **Speech** object to play back the contents of a string. In the following example, Microsoft Excel plays back "Hello". This example assumes speech features have been installed on the host system.

```
Sub UseSpeech()  
    Application.Speech.Speak "Hello"  
End Sub()
```

**Note** There is a speech feature in the setup tree that pertains to Dictation and Command & Control that does not have to be installed.



# SpellingOptions Object

[Application](#) | [SpellingOptions](#)

Represents the various spell checking options for a worksheet.

## Using the SpellingOptions object

Use the [SpellingOptions](#) property of the [Application](#) object to return a **SpellingOptions** object.

Once a **SpellingOptions** object is returned, you can use its following properties to set or return various spell checking options.

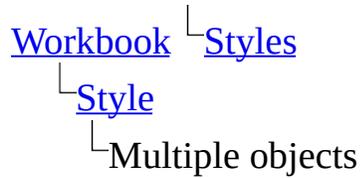
- [ArabicModes](#)
- [DictLang](#)
- [GermanPostReform](#)
- [HebrewModes](#)
- [IgnoreCaps](#)
- [IgnoreFileNames](#)
- [IgnoreMixedDigits](#)
- [KoreanCombineAux](#)
- [KoreanProcessCompound](#)
- [KoreanUseAutoChangeList](#)
- [SuggestMainOnly](#)
- [UserDict](#)

The following example uses the **IgnoreCaps** property to disable spell checking for words that have all capitalized letters. In this example, "Testt", but not "TESTT", is identified by the spell checker.

```
Sub IgnoreAllCAPS()  
  
    ' Place misspelled versions of the same word in all caps and mixe  
    Range("A1").Formula = "Testt"  
    Range("A2").Formula = "TESTT"  
  
    With Application.SpellingOptions  
        .SuggestMainOnly = True  
        .IgnoreCaps = True  
    End With  
  
    ' Run a spell check.  
    Cells.CheckSpelling  
  
End Sub
```



# Style Object



Represents a style description for a range. The **Style** object contains all style attributes (font, number format, alignment, and so on) as properties. There are several built-in styles, including Normal, Currency, and Percent. Using the **Style** object is a fast and efficient way to change several cell-formatting properties on multiple cells at the same time.

For the **Workbook** object, the **Style** object is a member of the **Styles** collection. The **Styles** collection contains all the defined styles for the workbook.

## Using the Style Object

Use the **Style** property to return the **Style** object used with a **Range** object. The following example applies the Percent style to cells A1:A10 on Sheet1.

```
Worksheets("Sheet1").Range("A1:A10").Style = "Percent"
```

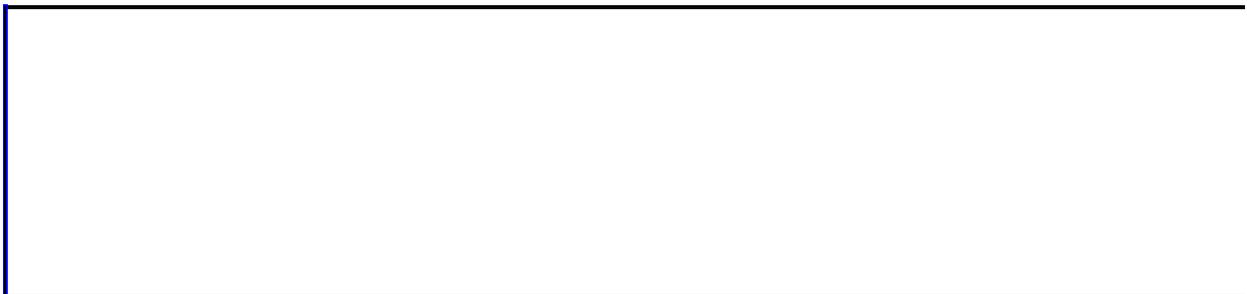
You can change the appearance of a cell by changing properties of the style applied to that cell. Keep in mind, however, that changing a style property will affect all cells already formatted with that style.

Use **Styles(index)**, where *index* is the style index number or name, to return a single **Style** object from the workbook **Styles** collection. The following example changes the Normal style for the active workbook by setting the style's **Bold** property.

```
ActiveWorkbook.Styles("Normal").Font.Bold = True
```

Styles are sorted alphabetically by style name. The style index number denotes the position of the specified style in the sorted list of style names. `Styles(1)` is the first style in the alphabetic list, and `Styles(Styles.Count)` is the last one in the list.

For more information about creating and modifying a style, see the [Styles](#) object.



# Tab Object

Multiple objects [L-Tab](#)

Represents a tab in a chart or a worksheet.

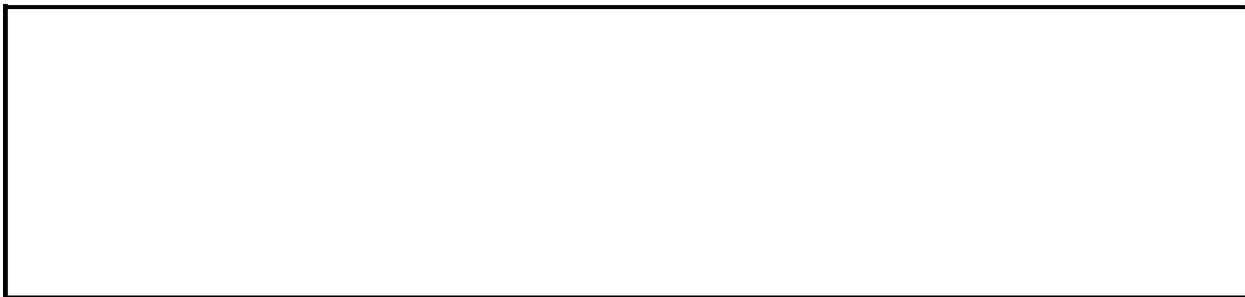
## Using the Tab object

Use the **Tab** property of the **Chart** object or **Worksheet** object to return a **Tab** object.

Once a **Tab** object is returned, you can use the **ColorIndex** property determine the settings of a tab for a chart or worksheet.

In the following example, Microsoft Excel determines if the worksheet's first tab color index is set to none and notifies the user.

```
Sub CheckTab()  
    ' Determine if color index of 1st tab is set to none.  
    If Worksheets(1).Tab.ColorIndex = xlColorIndexNone Then  
        MsgBox "The color index is set to none for the first " & _  
            "worksheet tab."  
    Else  
        MsgBox "The color index for the tab of the first worksheet "  
            "is not set none."  
    End If  
End Sub
```



# TextEffectFormat Object

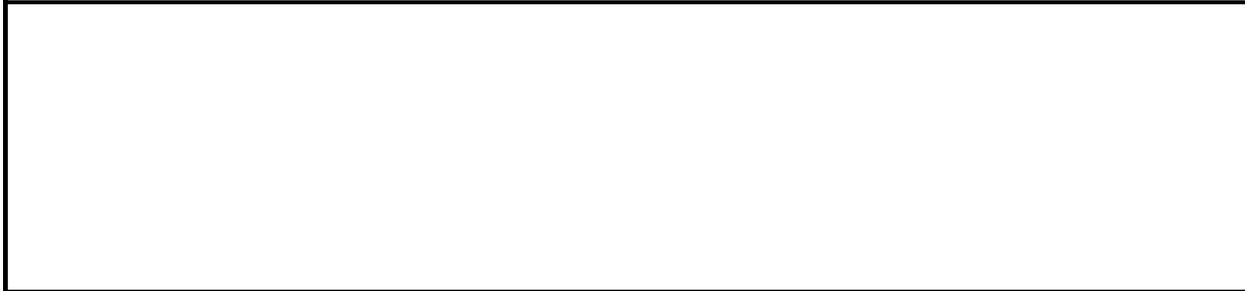
Multiple objects [↳TextEffectFormat](#)

Contains properties and methods that apply to WordArt objects.

## Using the TextEffectFormat Object

Use the **TextEffect** property to return a **TextEffectFormat** object. The following example sets the font name and formatting for shape one on myDocument. For this example to work, shape one must be a WordArt object.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(1).TextEffect
    .FontName = "Courier New"
    .FontBold = True
    .FontItalic = True
End With
```



# TextFrame Object

Multiple objects [↳ TextFrame](#)

Represents the text frame in a **Shape** object. Contains the text in the text frame as well as the properties and methods that control the alignment and anchoring of the text frame.

## Using the TextFrame Object

Use the **TextFrame** property to return a **TextFrame** object. The following example adds a rectangle to myDocument, adds text to the rectangle, and then sets the margins for the text frame.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddShape(msoShapeRectangle, _
    0, 0, 250, 140).TextFrame
    .Characters.Text = "Here is some test text"
    .MarginBottom = 10
    .MarginLeft = 10
    .MarginRight = 10
    .MarginTop = 10
End With
```



# ThreeDFormat Object

Multiple objects [└ThreeDFormat](#)  
[└ColorFormat](#)

Represents a shape's three-dimensional formatting.

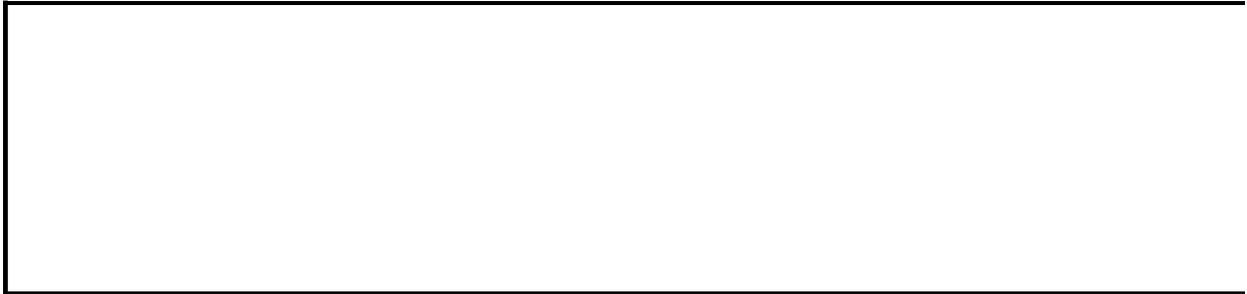
## Using The ThreeDFormat Object

Use the **ThreeD** property to return a **ThreeDFormat** object. The following example adds an oval to myDocument and then specifies that the oval be extruded to a depth of 50 points and that the extrusion be purple.

```
Set myDocument = Worksheets(1)
Set myShape = myDocument.Shapes.AddShape(msoShapeOval, _
    90, 90, 90, 40)
With myShape.ThreeD
    .Visible = True
    .Depth = 50
    .ExtrusionColor.RGB = RGB(255, 100, 255)
    ' RGB value for purple
End With
```

## Remarks

You cannot apply three-dimensional formatting to some kinds of shapes, such as beveled shapes or multiple-disjoint paths. Most of the properties and methods of the **ThreeDFormat** object for such a shape will fail.



# TickLabels Object

Multiple objects [└ TickLabels](#)  
[└ Font](#)

Represents the tick-mark labels associated with tick marks on a chart axis. This object isn't a collection. There's no object that represents a single tick-mark label; you must return all the tick-mark labels as a unit.

Tick-mark label text for the category axis comes from the name of the associated category in the chart. The default tick-mark label text for the category axis is the number that indicates the position of the category relative to the left end of this axis. To change the number of unlabeled tick marks between tick-mark labels, you must change the **TickLabelSpacing** property for the category axis.

Tick-mark label text for the value axis is calculated based on the **MajorUnit**, **MinimumScale**, and **MaximumScale** properties of the value axis. To change the tick-mark label text for the value axis, you must change the values of these properties.

## Using the TickLabels Object

Use the **TickLabels** property to return the **TickLabels** object. The following example sets the number format for the tick-mark labels on the value axis in embedded chart one on Sheet1.

```
Worksheets("sheet1").ChartObjects(1).Chart _  
    .Axes(xlValue).TickLabels.NumberFormat = "0.00"
```



[Show All](#)

# TreeviewControl Object

Represents the hierarchical member-selection control of a [cube](#) field. You use this object primarily for macro recording; it is not intended for any other use.

## Using the TreeviewControl Object

Use the [TreeviewControl](#) property to return the **TreeviewControl** object. The following example sets the control to its “drilled” (expanded, or visible) status for the states of California and Maryland in the second PivotTable report on the active worksheet.

```
ActiveSheet.PivotTables("PivotTable2") _  
    .CubeFields(1).TreeviewControl.Drilled = _  
        Array(Array("", ""), _  
              Array("[state].[states].[CA]", _  
                    "[state].[states].[MD]"))
```



# Trendline Object

[Trendlines](#) └ [Trendline](#)

└ Multiple objects

Represents a trendline in a chart. A trendline shows the trend, or direction, of data in a series. The **Trendline** object is a member of the [Trendlines](#) collection. The **Trendlines** collection contains all the **Trendline** objects for a single series.

## Using the Trendline Object

Use **Trendlines**(*index*), where *index* is the trendline index number, to return a single **Trendline** object. The following example changes the trendline type for the first series in embedded chart one on worksheet one. If the series has no trendline, this example will fail.

```
Worksheets(1).ChartObjects(1).Chart. _  
    SeriesCollection(1).Trendlines(1).Type = xlMovingAvg
```

The index number denotes the order in which the trendlines were added to the series. `Trendlines(1)` is the first trendline added to the series, and `Trendlines(Trendlines.Count)` is the last one added.



# UpBars Object

[ChartGroup](#) └ [UpBars](#)

└ Multiple objects

Represents the up bars in a chart group. Up bars connect points on series one with higher values on the last series in the chart group (the lines go up from series one). Only 2-D line groups that contain at least two series can have up bars. This object isn't a collection. There's no object that represents a single up bar; you either have up bars turned on for all points in a chart group or you have them turned off.

## Using the UpBars Object

Use the **UpBars** property to return the **UpBars** object. The following example turns on up and down bars for chart group one in embedded chart one on Sheet5. The example then sets the up bar color to blue and sets the down bar color to red.

```
With Worksheets("sheet5").ChartObjects(1).Chart.ChartGroups(1)
    .HasUpDownBars = True
    .UpBars.Interior.Color = RGB(0, 0, 255)
    .DownBars.Interior.Color = RGB(255, 0, 0)
End With
```

## Remarks

If the [HasUpDownBars](#) property is **False**, most properties of the **UpBars** object are disabled.

# UserAccess Object

[AllowEditRange](#) └ [UserAccessList](#)  
└ [UserAccess](#)

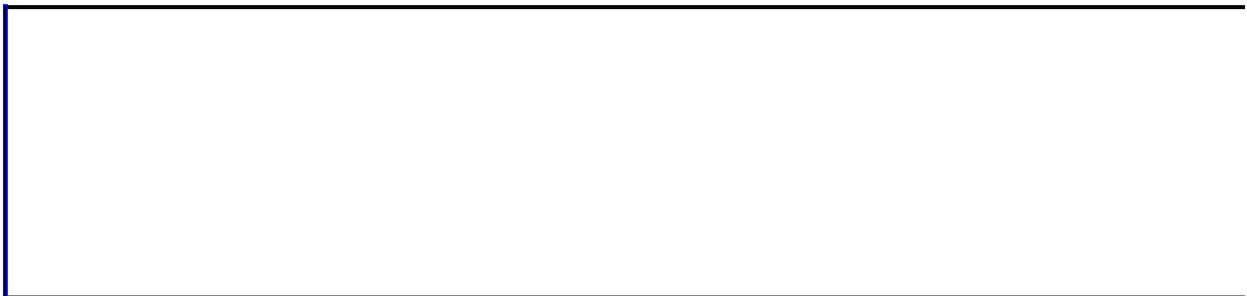
Represents the user access for a protected range.

## Using the **UserAccess** object

Use the [Add](#) method or the [Item](#) property of the [UserAccessList](#) collection to return a **UserAccess** object.

Once a **UserAccess** object is returned, you can determine if access is allowed for a particular range in an worksheet, using the [AllowEdit](#) property. The following example adds a range that can be edited on a protected worksheet and notifies the user the title of that range.

```
Sub UseAllowEditRanges()  
    Dim wksSheet As Worksheet  
  
    Set wksSheet = Application.ActiveSheet  
  
    ' Add a range that can be edited on the protected worksheet.  
    wksSheet.Protection.AllowEditRanges.Add "Test", Range("A1")  
  
    ' Notify the user the title of the range that can be edited.  
    MsgBox wksSheet.Protection.AllowEditRanges(1).Title  
  
End Sub
```



# Validation Object

[Range](#) | [Validation](#)

Represents data validation for a worksheet range.

## Using the Validation Object

Use the [Validation](#) property to return the **Validation** object. The following example changes the data validation for cell E5.

```
Range("e5").Validation _  
    .Modify xlValidateList, xlValidAlertStop, "=$A$1:$A$10"
```

Use the [Add](#) method to add data validation to a range and create a new **Validation** object. The following example adds data validation to cell E5.

```
With Range("e5").Validation  
    .Add Type:=xlValidateWholeNumber, _  
        AlertStyle:=xlValidAlertInformation, _  
        Minimum:="5", Maximum:="10"  
    .InputTitle = "Integers"  
    .ErrorTitle = "Integers"  
    .InputMessage = "Enter an integer from five to ten"  
    .ErrorMessage = "You must enter a number from five to ten"  
End With
```



# VPageBreak Object

Multiple objects [└ VPageBreaks](#)

[└ VPageBreak](#)

[└ Multiple objects](#)

Represents a vertical page break. The **VPageBreak** object is a member of the [VPageBreaks](#) collection.

## Using the VPageBreak Object

Use **VPageBreaks**(*index*), where *index* is the page break index number of the page break, to return a **VPageBreak** object. The following example changes the location of vertical page break one.

```
Worksheets(1).VPageBreaks(1).Location = Worksheets(1).Range("e5")
```



# Walls Object

[Chart](#) └ [Walls](#)

└ Multiple objects

Represents the walls of a 3-D chart. This object isn't a collection. There's no object that represents a single wall; you must return all the walls as a unit.

## Using the Walls Object

Use the **Walls** property to return the **Walls** object. The following example sets the pattern on the walls for embedded chart one on Sheet1. If the chart isn't a 3-D chart, this example will fail.

```
Worksheets("Sheet1").ChartObjects(1).Chart _  
    .Walls.Interior.Pattern = xlGray75
```



# Watch Object

[Application](#) └ [Watches](#)  
└ [Watch](#)

Represents a range which is tracked when the worksheet is recalculated. The **Watch** object allows users to verify the accuracy of their models and debug problems they encounter. The **Watch** object is a member of the [Watches](#) collection.

## Using the Watch object

Use the use the **Add** method or the **Item** property of the **Watches** collection to return a **Watch** object.

In the following example, Microsoft Excel creates a new **Watch** object using the **Add** method. This example creates a summation formula in cell A3, and then adds this cell to the watch facility.

```
Sub AddWatch()  
  
    With Application  
        .Range("A1").Formula = 1  
        .Range("A2").Formula = 2  
        .Range("A3").Formula = "=Sum(A1:A2)"  
        .Range("A3").Select  
        .Watches.Add Source:=ActiveCell  
    End With  
  
End Sub
```

You can specify to remove individual cells from the watch facility by using the **Delete** method of the **Watches** collection. This example deletes cell A3 on worksheet 1 of book 1 from the Watch Window. This example assumes you have added the cell A3 on sheet 1 of book 1 (using the previous example to add a **Watch** object).

```
Sub DeleteAWatch()  
  
    Application.Watches(Workbooks("Book1").Sheets("Sheet1").Range("A3"))  
  
End Sub
```

You can also specify to remove all cells from the Watch Window, by using the **Delete** method of the **Watches** collection. This example deletes all cells from the Watch Window.

```
Sub DeleteAllWatches()  
  
    Application.Watches.Delete
```

End Sub



# WebOptions Object

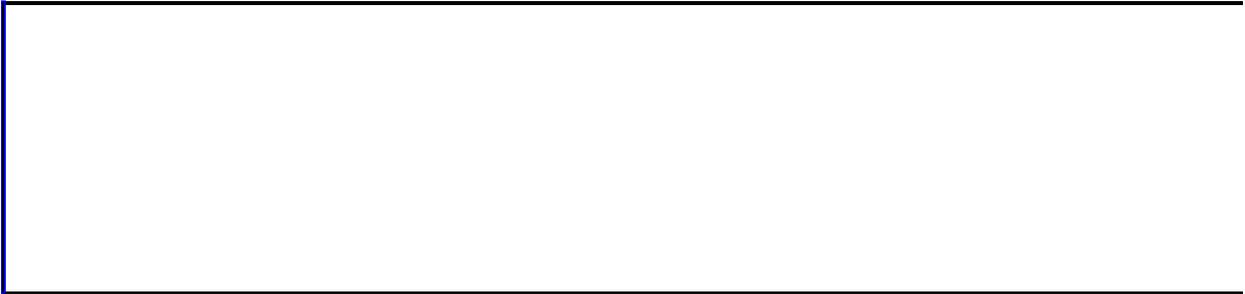
[Workbook](#)  [WebOptions](#)

Contains workbook-level attributes used by Microsoft Excel when you save a document as a Web page or open a Web page. You can return or set attributes either at the application (global) level or at the workbook level. (Note that attribute values can be different from one workbook to another, depending on the attribute value at the time the workbook was saved.) Workbook-level attribute settings override application-level attribute settings. Application-level attributes are contained in the [DefaultWebOptions](#) object.

## Using the WebOptions Object

Use the [WebOptions](#) property to return the **WebOptions** object. The following example checks to see whether PNG (Portable Network Graphics) is allowed as an image format and then sets the strImageFileType variable accordingly.

```
Set objAppWebOptions = Workbooks(1).WebOptions
With objAppWebOptions
    If .AllowPNG = True Then
        strImageFileType = "PNG"
    Else
        strImageFileType = "JPG"
    End If
End With
```



# Window Object

Multiple objects [Window](#)  
└─Multiple objects

Represents a window. Many worksheet characteristics, such as scroll bars and gridlines, are actually properties of the window. The **Window** object is a member of the [Windows](#) collection. The **Windows** collection for the **Application** object contains all the windows in the application, whereas the **Windows** collection for the **Workbook** object contains only the windows in the specified workbook.

## Using the Window Object

Use **Windows**(*index*), where *index* is the window name or index number, to return a single **Window** object. The following example maximizes the active window.

```
Windows(1).WindowState = xlMaximized
```

Note that the active window is always `windows(1)`.

The window caption is the text shown in the title bar at the top of the window when the window isn't maximized. The caption is also shown in the list of open files on the bottom of the **Windows** menu. Use the [Caption](#) property to set or return the window caption. Changing the window caption doesn't change the name of the workbook. The following example turns off cell gridlines for the worksheet shown in the Book1.xls:1 window.

```
Windows("book1.xls":1).DisplayGridlines = False
```



# Workbook Object

Multiple objects [Workbook](#)

└ Multiple objects

Represents a Microsoft Excel workbook. The **Workbook** object is a member of the [Workbooks](#) collection. The **Workbooks** collection contains all the **Workbook** objects currently open in Microsoft Excel.

## Using the Workbook Object

The following properties for returning a **Workbook** object are described in this section:

- **Workbooks** property
- **ActiveWorkbook** property
- **ThisWorkbook** property

## Workbooks Property

Use **Workbooks**(*index*), where *index* is the workbook name or index number, to return a single **Workbook** object. The following example activates workbook one.

```
Workbooks(1).Activate
```

The index number denotes the order in which the workbooks were opened or created. `workbooks(1)` is the first workbook created, and `workbooks(workbooks.Count)` is the last one created. Activating a workbook doesn't change its index number. All workbooks are included in the index count, even if they're hidden.

The **Name** property returns the workbook name. You cannot set the name by using this property; if you need to change the name, use the **SaveAs** method to save the workbook under a different name. The following example activates Sheet1 in the workbook named Cogs.xls (the workbook must already be open in Microsoft Excel).

```
Workbooks("Cogs.xls").Worksheets("Sheet1").Activate
```

## ActiveWorkbook Property

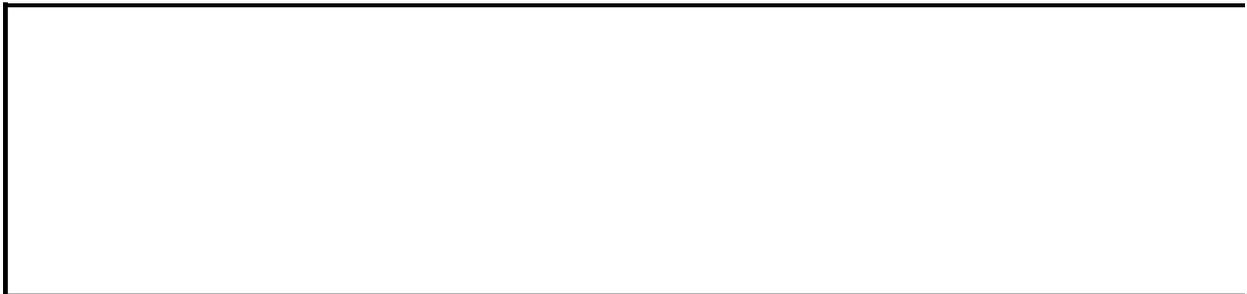
The **ActiveWorkbook** property returns the workbook that's currently active. The following example sets the name of the author for the active workbook.

```
ActiveWorkbook.Author = "Jean Selva"
```

## **ThisWorkbook Property**

The **ThisWorkbook** property returns the workbook where the Visual Basic code is running. In most cases, this is the same as the active workbook. However, if the Visual Basic code is part of an add-in, the **ThisWorkbook** property won't return the active workbook. In this case, the active workbook is the workbook calling the add-in, whereas the **ThisWorkbook** property returns the add-in workbook.

If you'll be creating an add-in from your Visual Basic code, you should use the **ThisWorkbook** property to qualify any statement that must be run on the workbook you compile into the add-in.



# Worksheet Object

Multiple objects  [Worksheet](#)

 Multiple objects

Represents a worksheet. The **Worksheet** object is a member of the [Worksheets](#) collection. The **Worksheets** collection contains all the **Worksheet** objects in a workbook.

## Using the Worksheet Object

The following properties for returning a **Worksheet** object are described in this section:

- **Worksheets** property
- **ActiveSheet** property

## Worksheets Property

Use **Worksheets**(*index*), where *index* is the worksheet index number or name, to return a single **Worksheet** object. The following example hides worksheet one in the active workbook.

```
Worksheets(1).Visible = False
```

The worksheet index number denotes the position of the worksheet on the workbook's tab bar. `worksheets(1)` is the first (leftmost) worksheet in the workbook, and `worksheets(Worksheets.Count)` is the last one. All worksheets are included in the index count, even if they're hidden.

The worksheet name is shown on the tab for the worksheet. Use the [Name](#) property to set or return the worksheet name. The following example protects the scenarios on Sheet1.

```
Dim strPassword As String  
strPassword = InputBox ("Enter the password for the worksheet")  
Worksheets("Sheet1").Protect password:=strPassword, scenarios:=True
```

The **Worksheet** object is also a member of the [Sheets](#) collection. The **Sheets** collection contains all the sheets in the workbook (both chart sheets and worksheets).

## ActiveSheet Property

When a worksheet is the active sheet, you can use the **ActiveSheet** property to refer to it. The following example uses the **Activate** method to activate Sheet1, sets the page orientation to landscape mode, and then prints the worksheet.

```
Worksheets("Sheet1").Activate  
ActiveSheet.PageSetup.Orientation = xlLandscape  
ActiveSheet.PrintOut
```



# WorksheetFunction Object

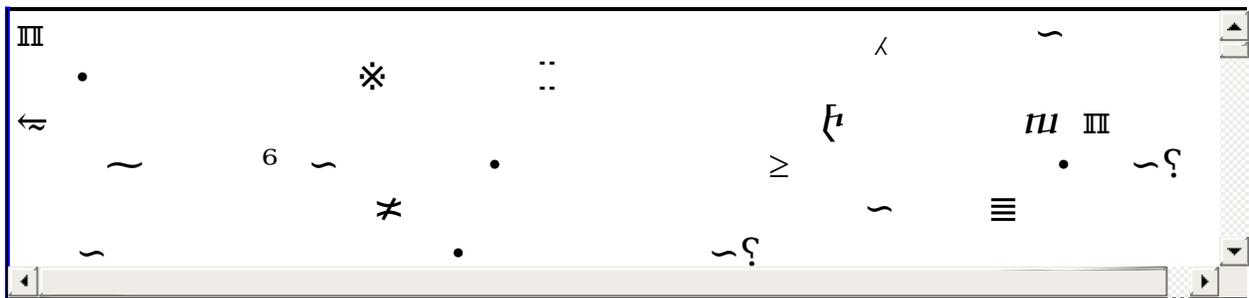
[Application](#) `WorksheetFunction`

Used as a container for Microsoft Excel worksheet functions that can be called from Visual Basic.

## Using the WorksheetFunction Object

Use the [WorksheetFunction](#) property to return the **WorksheetFunction** object. The following example displays the result of applying the **Min** worksheet function to the range A1:A10.

```
Set myRange = Worksheets("Sheet1").Range("A1:C10")  
answer = Application.WorksheetFunction.Min(myRange)  
MsgBox answer
```



# XmlDataBinding Object

[XmlMap](#)  [XmlDataBinding](#)

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Represents the connection to the source data for an **XmlMap** object.

## Using the XmlDataBinding Object

Use the **LoadSettings** method initialize the settings for an **XmlDataBinding** object

Use the **Refresh** method to refresh a data binding.

Use the **ClearSettings** method to remove a data binding.



# XmlMap Object

Multiple objects [XmlMap](#)  
└─Multiple objects

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Represents an XML map that has been added to a workbook.

## Using the XmlMap Object

Use the **Add** method of the [XmlMaps](#) collection to add an XML map to a workbook.

### Importing and exporting XML data

Use the [Import](#) method to import XML data from an XML data file into cells mapped to the specified **XmlMap**. The [ImportXml](#) method imports XML data for a **String** variable.

Use the [Export](#) method to export data from cells mapped to the specified **XmlMap**. The [ExportXml](#) method exports XML data to a **String** variable.



# XmlNamespace Object

Multiple objects [L-XmlNamespace](#)

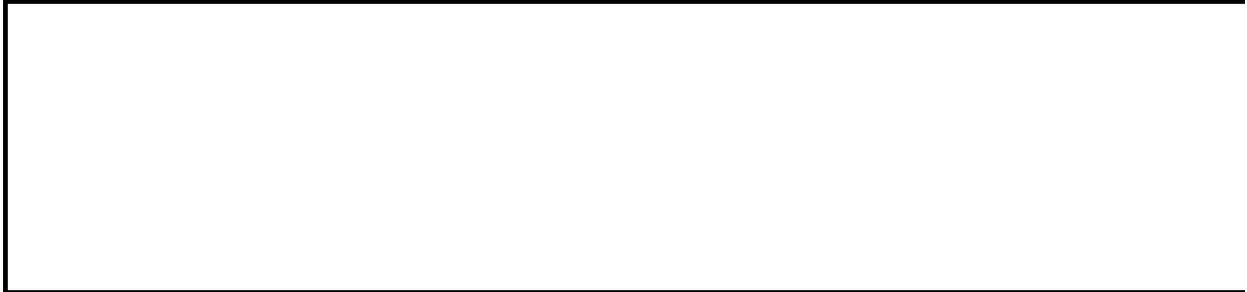
**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Represents a namespace that has been added to a workbook.

## Using the XmlNamespace Object

Use the **Prefix** property to return the prefix of an **XmlNamespace** object.

Use the **Uri** property to return the Uniform Resource Identifier (URI) of an **XmlNamespace** object.



# XmlSchema Object

[XmlMap](#) └ [XmlSchemas](#)  
└ [XmlSchema](#)  
└ [XmlNamespace](#)

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Represents an XML schema contained by an **XmlMap** object.

## Using the XmlSchema Object

Use the **Item** method to return an **XmlSchema** object from the **XmlSchemas** collection.

Use the **Namespace** property to return the target namespace for a schema.

Use the **XML** property to return the XML contents of a schema.



# XPath Object

Multiple objects [XPath](#)  
[XmlMap](#)

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Represents an XPath that has been mapped to a **Range** or **ListColumn** object.

## Using the XPath Object

Use the **SetValue** method to map an XPath to a range or list column. The **SetValue** method is also used to change the properties of an existing XPath.

The following example creates an XML list based on the "Contacts" schema map that is attached to the workbook, then uses the **SetValue** method to bind each column to an XPath.

```
Sub CreateXMLList()  
    Dim mapContact As XmlMap  
    Dim strXPath As String  
    Dim lstContacts As ListObject  
    Dim lcNewCol As ListColumn  
  
    ' Specify the schema map to use.  
    Set mapContact = ActiveWorkbook.XmlMaps("Contacts")  
  
    ' Create a new list.  
    Set lstContacts = ActiveSheet.ListObjects.Add  
  
    ' Specify the first element to map.  
    strXPath = "/Root/Person/FirstName"  
    ' Map the element.  
    lstContacts.ListColumns(1).XPath.SetValue mapContact, strXPath  
  
    ' Specify the element to map.  
    strXPath = "/Root/Person/LastName"  
    ' Add a column to the list.  
    Set lcNewCol = lstContacts.ListColumns.Add  
    ' Map the element.  
    lcNewCol.XPath.SetValue mapContact, strXPath  
  
    strXPath = "/Root/Person/Address/Zip"  
    Set lcNewCol = lstContacts.ListColumns.Add  
    lcNewCol.XPath.SetValue mapContact, strXPath  
End Sub
```

Use the **Clear** method to remove an XPath that has been mapped to a range or list column.



# AcceptAllChanges Method

Accepts all changes in the specified shared workbook.

*expression*.AcceptAllChanges(*When, Who, Where*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**When** Optional **Variant**. Specifies when all the changes are accepted.

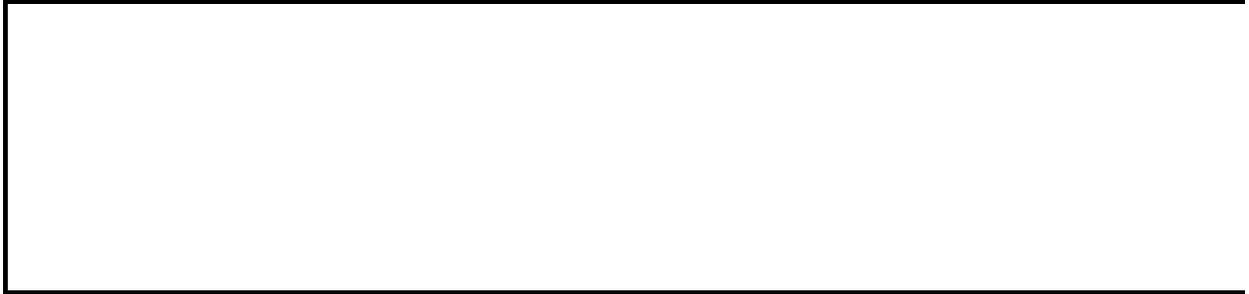
**Who** Optional **Variant**. Specifies by whom all the changes are accepted.

**Where** Optional **Variant**. Specifies where all the changes are accepted.

## Example

This example accepts all changes in the active workbook.

`ActiveWorkbook.AcceptAllChanges`



[Show All](#)

# Activate Method

[Activate method as it applies to the \*\*Chart\*\* and \*\*ChartObject\*\* object.](#)

Makes the current chart the active chart.

*expression*.**Activate**

*expression* Required. An expression that returns one of the above objects.

[Activate method as it applies to the \*\*Worksheet\*\* object.](#)

Makes the current sheet the active sheet. Equivalent to clicking the sheet's tab.

*expression*.**Activate**

*expression* Required. An expression that returns one of the above objects.

[Activate method as it applies to the \*\*OLEObject\*\* object.](#)

Activates the object.

*expression*.**Activate**

*expression* Required. An expression that returns one of the above objects.

[Activate method as it applies to the \*\*Pane\*\* object.](#)

Activates the pane. If the pane isn't in the active window, the window that the pane belongs to will also be activated. You cannot activate a frozen pane.

*expression*.**Activate**

*expression* Required. An expression that returns one of the above objects.

[Activate method as it applies to the \*\*Range\*\* object.](#)

Activates a single cell, which must be inside the current selection. To select a range of cells, use the **Select** method.

*expression*.**Activate**

*expression* Required. An expression that returns one of the above objects.

[Activate method as it applies to the \*\*Window\*\* object.](#)

Brings the window to the front of the z-order. This won't run any Auto\_Activate or Auto\_Deactivate macros that might be attached to the workbook (use the **RunAutoMacros** method to run those macros).

*expression*.**Activate**

*expression* Required. An expression that returns one of the above objects.

[Activate method as it applies to the \*\*Workbook\*\* object.](#)

Activates the first window associated with the workbook. This won't run any Auto\_Activate or Auto\_Deactivate macros that might be attached to the workbook (use the **RunAutoMacros** method to run those macros).

*expression*.**Activate**

*expression* Required. An expression that returns one of the above objects.

## Example

[As it applies to the \*\*Worksheet\*\* object.](#)

This example activates Sheet1.

```
Worksheets("Sheet1").Activate
```

[As it applies to the \*\*Range\*\* object.](#)

This example selects cells A1:C3 on Sheet1 and then makes cell B2 the active cell.

```
Worksheets("Sheet1").Activate  
Range("A1:C3").Select  
Range("B2").Activate
```

[As it applies to the \*\*Workbook\*\* object.](#)

This example activates Book4.xls. If Book4.xls has multiple windows, the example activates the first window, Book4.xls:1.

```
Workbooks("BOOK4.XLS").Activate
```

[Show All](#)

# ActivateMicrosoftApp Method

Activates a Microsoft application. If the application is already running, this method activates the running application. If the application isn't running, this method starts a new instance of the application.

*expression*.**ActivateMicrosoftApp**(*index*)

*expression* Required. An expression that returns an **Application** object.

*index* Required [XlMSApplication](#). Specifies the Microsoft application to activate.

XlMSApplication can be one of these XlMSApplication constants.

**xlMicrosoftWord**

**xlMicrosoftPowerPoint**

**xlMicrosoftMail**

**xlMicrosoftAccess**

**xlMicrosoftFoxPro**

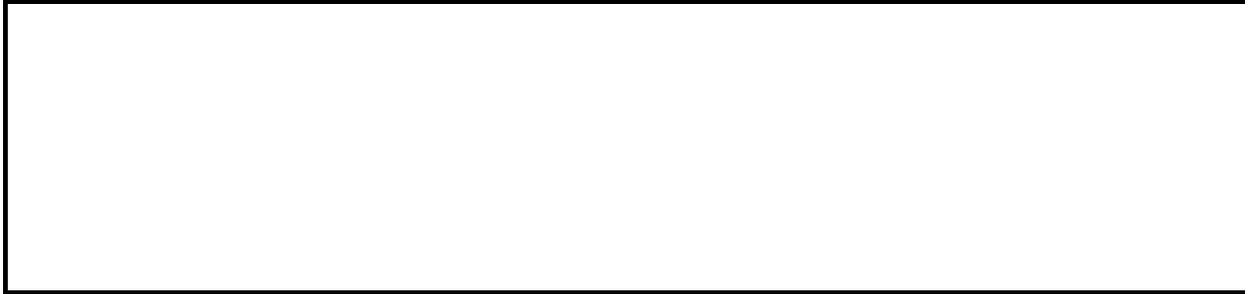
**xlMicrosoftProject**

**xlMicrosoftSchedulePlus**

## Example

This example starts and activates Word.

```
Application.ActivateMicrosoftApp xlMicrosoftWord
```



# ActivateNext Method

Activates the specified window and then sends it to the back of the window z-order.

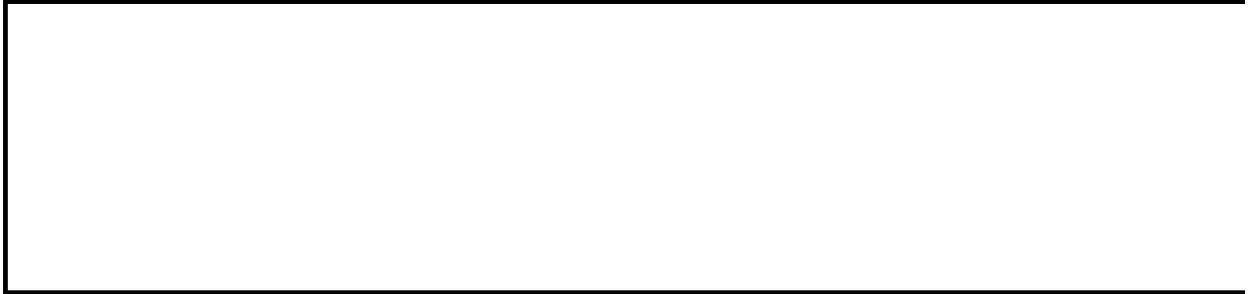
*expression*.**ActivateNext**

*expression* Required. An expression that returns a **Window** object.

## Example

This example sends the active window to the back of the z-order.

`ActiveWindow.ActivateNext`



# ActivatePrevious Method

Activates the specified window and then activates the window at the back of the window z-order.

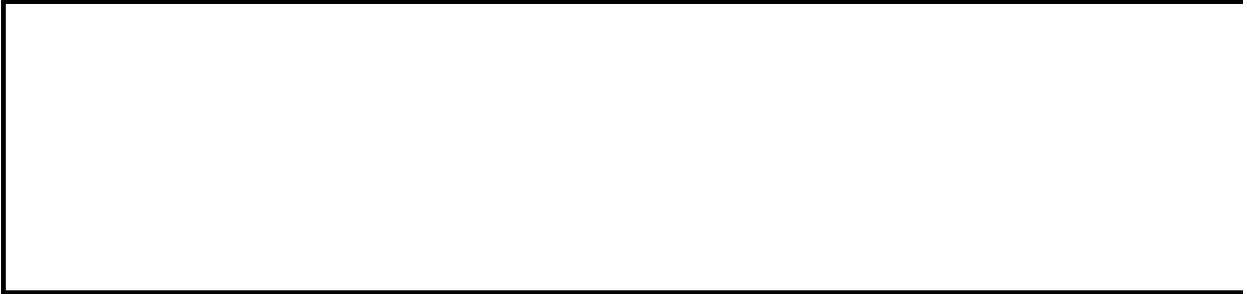
*expression*.**ActivatePrevious**

*expression* Required. An expression that returns a **Window** object.

## Example

This example activates the window at the back of the z-order.

`ActiveWindow.ActivatePrevious`



[Show All](#)

# Add Method

 [Add method as it applies to the \*\*AddIns\*\* object.](#)

Adds a new add-in file to the list of add-ins. Returns an [AddIn](#) object.

*expression*.**Add**(*FileName*, *CopyFile*)

*expression* Required. An expression that returns an **AddIns** object.

**Filename** Required **String**. The name of the file that contains the add-in you want to add to the list in the add-in manager.

**CopyFile** Optional **Variant**. Ignored if the add-in file is on a hard disk. **True** to copy the add-in to your hard disk, if the add-in is on a removable medium (a floppy disk or compact disc). **False** to have the add-in remain on the removable medium. If this argument is omitted, Microsoft Excel displays a dialog box and asks you to choose.

## Remarks

This method doesn't install the new add-in. You must set the [Installed](#) property to install the add-in.

[Add method as it applies to the \*\*AllowEditRanges\*\* object.](#)

Adds a range that can be edited on a protected worksheet. Returns a [AllowEditRange](#) object.

*expression*.**Add(Title, Range, Password)**

*expression* Required. An expression that returns an **AllowEditRanges** object.

**Title** Required **String**. The title of range.

**Range** Required **Range** object. The range allowed to be edited.

**Password** Optional **Variant**. The password for the range.

[Add method as it applies to the \*\*CalculatedFields\*\* object.](#)

Creates a new calculated field. Returns a [PivotField](#) object.

*expression*.**Add(Name, Formula, UseStandardFormula)**

*expression* Required. An expression that returns a **CalculatedFields** object.

**Name** Required **String**. The name of the field.

**Formula** Required **String**. The formula for the field.

**UseStandardFormula** Optional **Variant**. **False** (default) for upward compatibility. **True** for strings contained in any arguments that are field names, will be interpreted as having been formatted in standard U.S. English instead of local settings.

[Add method as it applies to the \*\*CalculatedItems\*\* object.](#)

Creates a new calculated item. Returns a [PivotItem](#) object.

*expression.Add(Name, Formula, UseStandardFormula)*

*expression* Required. An expression that returns a **CalculatedItems** object.

**Name** Required **String**. The name of the item.

**Formula** Required **String**. The formula for the item.

**UseStandardFormula** Optional **Variant**. **False** (default) for upward compatibility. **True** for strings contained in any arguments that are item names, will be interpreted as having been formatted in standard U.S. English instead of local settings.



[Add method as it applies to the \*\*CalculatedMembers\*\* object.](#)

Adds a calculated field or calculated item to a PivotTable. Returns a [CalculatedMember](#) object.

*expression.Add(Name, Formula, SolveOrder, Type)*

*expression* Required. An expression that returns a **CalculatedMembers** object.

**Name** Required **String**. The name of the calculated member.

**Formula** Required **String**. The formula of the calculated member.

**SolveOrder** Optional **Variant**. The solve order for the calculated member.

**Type** Optional **Variant**. The type of calculated member.

## Remarks

The **Formula** argument must have a valid MDX (Multidimensional Expression) syntax statement. The **Name** argument has to be acceptable to the [Online Analytical Processing \(OLAP\)](#) provider and the **Type** argument has to be defined.

If you set the **Type** argument of this method to **xlCalculatedSet**, then you must call the [AddSet](#) method to make the new field set visible in the PivotTable.

[Add method as it applies to the \*\*ChartObjects\*\* object.](#)

Creates a new embedded chart. Returns a [ChartObject](#) object.

*expression.Add(Left, Top, Width, Height)*

*expression* Required. An expression that returns a **ChartObjects** object.

**Left**, **Top** Required **Double**. The initial coordinates of the new object (in points), relative to the upper-left corner of cell A1 on a worksheet or to the upper-left corner of a chart.

**Width**, **Height** Required **Double**. The initial size of the new object, in points.

[Add method as it applies to the \*\*Charts\*\* object.](#)

Creates a new chart sheet. Returns a [Chart](#) object.

*expression.Add(Before, After, Count)*

*expression* Required. An expression that returns a **Charts** object.

**Before** Optional **VARIANT**. An object that specifies the sheet before which the new sheet is added.

**After** Optional **VARIANT**. An object that specifies the sheet after which the new sheet is added.

**Count** Optional **Variant**. The number of sheets to be added. The default value is one.

## Remarks

If **Before** and **After** are both omitted, the new chart is inserted before the active sheet.

[Add method as it applies to the \*\*CustomProperties\*\* object.](#)

Adds custom property information. Returns a **CustomProperty** object.

*expression.Add(Name, Value)*

*expression* Required. An expression that returns a **CustomProperties** object.

**Name** Required **String**. The name of the custom property.

**Value** Required **Variant**. The value of the custom property.

[Add method as it applies to the \*\*CustomViews\*\* object.](#)

Creates a new custom view. Returns a **CustomView** object that represents the new view.

*expression.Add(ViewName, PrintSettings, RowColSettings)*

*expression* Required. An expression that returns a **CustomViews** object.

**ViewName** Required **String**. The name of the new view.

**PrintSettings** Optional **Variant**. **True** to include print settings in the custom view.

**RowColSettings** Optional **Variant**. **True** to include settings for hidden rows and columns (including filter information) in the custom view.

[Add method as it applies to the \*\*FormatConditions\*\* object.](#)

Adds a new conditional format. Returns a **FormatCondition** object that represents the new conditional format.

*expression.Add(Type, Operator, Formula1, Formula2)*

*expression* Required. An expression that returns a **FormatConditions** object.

**Type** Required [XlFormatConditionType](#). Specifies whether the conditional format is based on a cell value or an expression.

XlFormatConditionType can be one of these XlFormatConditionType constants.

**xlCellValue** The conditional format is based on a cell value.

**xlExpression** The conditional format is based on an expression.

**Operator** Optional **Variant**. The conditional format operator. Can be one of the following **XlFormatConditionOperator** constants: **xlBetween**, **xlEqual**, **xlGreater**, **xlGreaterEqual**, **xlLess**, **xlLessEqual**, **xlNotBetween**, or **xlNotEqual**. If *Type* is **xlExpression**, the **Operator** argument is ignored.

**Formula1** Optional **Variant**. The value or expression associated with the conditional format. Can be a constant value, a string value, a cell reference, or a formula.

**Formula2** Optional **Variant**. The value or expression associated with the second part of the conditional format when **Operator** is **xlBetween** or **xlNotBetween** (otherwise, this argument is ignored). Can be a constant value, a string value, a cell reference, or a formula.

## Remarks

You cannot define more than three conditional formats for a range. Use the **Modify** method to modify an existing conditional format, or use the **Delete** method to delete an existing format before adding a new one.

 [Add method as it applies to the \*\*HPageBreaks\*\* object.](#)

Adds a horizontal page break. Returns an **HPageBreak** object.

*expression*.**Add(Before)**

*expression* Required. An expression that returns an **HPageBreaks** object.

**Before** Required **Object**. A **Range** object. The range above which the new page break will be added.

 [Add method as it applies to the \*\*Hyperlinks\*\* object.](#)

Adds a hyperlink to the specified range or shape. Returns a **Hyperlink** object.

*expression*.**Add(Anchor, Address, SubAddress, ScreenTip, TextToDisplay)**

*expression* Required. An expression that returns a **Hyperlinks** object.

**Anchor** Required **Object**. The anchor for the hyperlink. Can be either a **Range** or **Shape** object.

**Address** Required **String**. The address of the hyperlink.

**SubAddress** Optional **VARIANT**. The subaddress of the hyperlink.

**ScreenTip** Optional **VARIANT**. The screen tip to be displayed when the mouse pointer is paused over the hyperlink.

**TextToDisplay** Optional **VARIANT**. The text to be displayed for the hyperlink.

## Remarks

When you specify the *TextToDisplay* argument, the text must be a string.

[Add method as it applies to the \*\*ListColumns\*\* collection object.](#)

Adds a new column to the list object. Returns a **ListColumn** object.

*expression*.**Add(Position)**

*expression* Required. An expression that returns a **ListColumns** object for the newly created column.

**Position** Optional **Integer**. Specifies the relative position of the new column that starts at 1. The previous column at this position is shifted outward.

## Remarks

If **Position** is not specified, a new rightmost column is added. A name for the column is automatically generated. The name of the new column can be changed after the column is added.

 [Add method as it applies to the \*\*ListObjects\*\* collection object.](#)

Creates a new list object. Returns a **ListObject** object.

*expression*.**Add(SourceType, Source, LinkSource, HasHeaders, Destination)**

*expression* Required. An expression that returns a **ListObjects** object.

**SourceType** Optional **XIListObjectSourceType**. Indicates the kind of source for the query. Can be one of the following **XIListObjectSourceType** constants: **xlSrcExternal**, or **xlSrcRange**. If omitted, the **SourceType** will default to **xlSrcRange**.

**Source** Optional when **SourceType** = **xlSrcRange**. A **Range** object representing the data source. If omitted, the **Source** will default to the range returned by list range detection code. Required when **SourceType** = **xlSrcExternal**. An array of **String** values specifying a connection to the source.

Element#	Contents
0	URL to SharePoint Service
1	ListName
2	ViewGUID

**LinkSource** Optional **Boolean**. Indicates whether an external data source is to be linked to the **ListObject** object. If **SourceType** is **xlSrcExternal**, default is **True**. Invalid if **SourceType** is **xlSrcRange**, and will return an error if not omitted.

**HasHeaders** Optional **Variant**. An [XIYesNoGuess](#) constant that indicates whether the data being imported has column labels. If the **Source** does not contain headers, Excel will automatically generate headers.

HasHeaders can be one of these **XIYesNoGuess** constants.

**xlGuess**

**xlNo**

**xlYes**

**Destination** Optional **Variant**. A **Range** object specifying a single-cell reference as the destination for the top-left corner of the new list object. If the **Range** object refers to more than one cell, an error is generated. The **Destination** argument must be specified when **SourceType** is set to **xlSrcExternal**. The **Destination** argument is ignored if **SourceType** is set to **xlSrcRange**. The destination range must be on the worksheet that contains the **ListObjects** collection specified by *expression*. New columns will be inserted at the **Destination** to fit the new list. Therefore, existing data will not be overwritten.

## Remarks

When the list has headers, the first row of cells will be converted to **Text**, if not already set to text. The conversion will be based on the visible text for the cell. This means that if there is a date value with a **Date** format that changes with locale, the conversion to a list might produce different results depending on the current system locale. Moreover, if there are two cells in the header row that have the same visible text, an incremental **Integer** will be appended to make each column header unique.

 [Add method as it applies to the \*\*ListRows\*\* collection object.](#)

Adds a new row to the list object. Returns a **ListRow** object.

*expression*.**Add(Position)**

*expression* Required. An expression that returns a **ListRows** object for the newly created row.

**Position** Optional **Integer**. Specifies the relative position of the new row.

## Remarks

If **Position** is not specified, a new bottom row is added.



[Add method as it applies to the Names object.](#)

Defines a new name. Returns a **Name** object.

**expression.Add(Name, RefersTo, Visible, MacroType, ShortcutKey, Category, NameLocal, RefersToLocal, CategoryLocal, RefersToR1C1, RefersToR1C1Local)**

**expression** Required. An expression that returns a **Names** object.

**Name** Optional **Variant**. Required if **NameLocal** isn't specified. The text to use as the name (in the language of the macro). Names cannot include spaces and cannot look like cell references.

**RefersTo** Optional **Variant**. Required unless one of the other **RefersTo** arguments is specified. Describes what the name refers to (in the language of the macro, using A1-style notation). **Note** Nothing is returned if the reference does not exist.

**Visible** Optional **Variant**. **True** to define the name normally. **False** to define the name as a hidden name (that is, it doesn't appear in either the **Define Name**, **Paste Name**, or **Goto** dialog box). The default value is **True**.

**MacroType** Optional **Variant**. The macro type, as shown in the following table.

Value	Meaning
1	User-defined function ( <b>Function</b> procedure)
2	Macro (also known as <b>Sub</b> procedure)
3 or omitted	None (that is, the name doesn't refer to a user-defined function or macro)

**ShortcutKey** Optional **Variant**. The macro shortcut key. Must be a single letter, such as "z" or "Z". Applies only for command macros.

**Category** Optional **Variant**. The category of the macro or function if **MacroType** is 1 or 2. The category is used in the Function Wizard. Existing categories can be referred to either by number (starting at 1) or by name (in the language of the macro). Microsoft Excel creates a new category if the specified category doesn't already exist.

**NameLocal** Optional **Variant**. Required if **Name** isn't specified. The text to use as the name (in the language of the user). Names cannot include spaces and cannot look like cell references.

**RefersToLocal** Optional **Variant**. Required unless one of the other **RefersTo** arguments is specified. Describes what the name refers to (in the language of the user, using A1-style notation).

**CategoryLocal** Optional **Variant**. Required if **Category** isn't specified. Text identifying the category of a custom function in the language of the user.

**RefersToR1C1** Optional **Variant**. Required unless one of the other **RefersTo** arguments is specified. Describes what the name refers to (in the language of the macro, using R1C1-style notation).

**RefersToR1C1Local** Optional **Variant**. Required unless one of the other **RefersTo** arguments is specified. Describes what the name refers to (in the language of the user, using R1C1-style notation).

[Add method as it applies to the \*\*OLEObjects\*\* object.](#)

Adds a new OLE object to a sheet. Returns an **OLEObject** object.

*expression*.**Add(ClassType, FileName, Link, DisplayAsIcon, IconFileName, IconIndex, IconLabel, Left, Top, Width, Height)**

*expression* Required. An expression that returns an **OLEObjects** object.

**ClassType** Optional **Variant**. (you must specify either **ClassType** or **FileName**). A string that contains the programmatic identifier for the object to be created. If **ClassType** is specified, **FileName** and **Link** are ignored.

**FileName** Optional **Variant**. (you must specify either **ClassType** or **FileName**). A string that specifies the file to be used to create the OLE object.

**Link** Optional **VARIANT**. **True** to have the new OLE object based on **FileName** be linked to that file. If the object isn't linked, the object is created as a copy of the file. The default value is **False**.

**DisplayAsIcon** Optional **VARIANT**. **True** to display the new OLE object either as an icon or as its regular picture. If this argument is **True**, **IconFileName** and **IconIndex** can be used to specify an icon.

**IconFileName** Optional **VARIANT**. A string that specifies the file that contains the icon to be displayed. This argument is used only if **DisplayAsIcon** is **True**. If this argument isn't specified or the file contains no icons, the default icon for the OLE class is used.

**IconIndex** Optional **VARIANT**. The number of the icon in the icon file. This is used only if **DisplayAsIcon** is **True** and **IconFileName** refers to a valid file that contains icons. If an icon with the given index number doesn't exist in the file specified by **IconFileName**, the first icon in the file is used.

**IconLabel** Optional **VARIANT**. A string that specifies a label to display beneath the icon. This is used only if **DisplayAsIcon** is **True**. If this argument is omitted or is an empty string (""), no caption is displayed.

**Left** , **Top** Optional **VARIANT**. The initial coordinates of the new object, in points, relative to the upper-left corner of cell A1 on a worksheet, or to the upper-left corner of a chart.

**Width** , **Height** Optional **VARIANT**. The initial size of the new object, in points.



[Add method as it applies to the \*\*Parameters\*\* object.](#)

Creates a new query parameter. Returns a **Parameter** object.

*expression*.**Add**(**Name**, **iDataType**)

*expression* Required. An expression that returns a **Parameters** object.

**Name** Required **String**. The name of the specified parameter. The parameter name should match the parameter clause in the SQL statement.

**iDataType** Optional **VARIANT**. The data type of the parameter. Can be any

[xlParameterDataType](#) constant:

<b>xlParamTypeBigInt</b>	<b>xlParamTypeNumeric</b>
<b>xlParamTypeBinary</b>	<b>xlParamTypeLongVarChar</b>
<b>xlParamTypeBit</b>	<b>xlParamTypeReal</b>
<b>xlParamTypeChar</b>	<b>xlParamTypeSmallInt</b>
<b>xlParamTypeDate</b>	<b>xlParamTypeTime</b>
<b>xlParamTypeDecimal</b>	<b>xlParamTypeTimeStamp</b>
<b>xlParamTypeDouble</b>	<b>xlParamTypeTinyInt</b>
<b>xlParamTypeFloat</b>	<b>xlParamTypeUnknown</b>
<b>xlParamTypeInteger</b>	<b>xlParamTypeVarBinary</b>
<b>xlParamTypeLongVarBinary</b>	<b>xlParamTypeVarChar</b>
<b>xlParamTypeWChar</b>	

These values correspond to ODBC data types. They indicate the type of value the ODBC driver is expecting to receive. Microsoft Excel and the ODBC driver manager will coerce the parameter value given in Microsoft Excel into the correct data type for the driver.

[Add method as it applies to the \*\*Phonetics\*\* object.](#)

Adds phonetic text to the specified cellt.

*expression*.**Add(Start, Length, Text)**

*expression* Required. An expression that returns a **Phonetics** object.

**Start** Required **Long**. The position that represents the first character in the specified cell.

**Length** Required **Long**. The number of characters from the **Start** position to the end of the text in the cell.

**Text** Required **String**. Collectively, the characters that represent the phonetic text in the cell.

[Add method as it applies to the \*\*PivotCaches\*\* object.](#)

Adds a new PivotTable cache to a **PivotCaches** collection. Returns a **PivotCache** object.

*expression.Add(SourceType, SourceData)*

*expression* Required. An expression that returns a **PivotCaches** object.

**SourceType** Required **XlPivotTableSourceType**. The source of the PivotTable cache data.

XlPivotTableSourceType can be one of these XlPivotTableSourceType constants.

**xlConsolidation**

**xlDatabase**

**xlExternal**

**xlPivotTable**

**xlScenario**

**SourceData** Optional **Variant**. The data for the new PivotTable cache. This argument is required if **SourceType** isn't **xlExternal**. Can be a **Range** object, an array of ranges, or a text constant that represents the name of an existing PivotTable report. For an external database, this is a two-element array. The first element is the connection string specifying the provider of the data. The second element is the SQL query string used to get the data. If you specify this argument, you must also specify **SourceType**.

## Remarks

If the PivotTable cache isn't referenced by a [PivotTable](#) object, the PivotTable cache is automatically deleted before the workbook is saved.

 [Add method as it applies to the \*\*PivotFormulas\*\* object.](#)

Creates a new PivotTable formula. Returns a [PivotFormula](#) object.

*expression*.**Add(Formula, UseStandardFormula)**

*expression* Required. An expression that returns one of the above objects.

**Formula** Required **String**. The new PivotTable formula.

**UseStandardFormula** Optional **Variant**. A standard PivotTable formula.

 [Add method as it applies to the \*\*PivotItems\*\* object.](#)

Creates a new PivotTable item.

*expression*.**Add(Name)**

*expression* Required. An expression that returns a **PivotItems** object.

**Name** Required **String**. The name of the new PivotTable item.

 [Add method as it applies to the \*\*PivotTables\*\* object.](#)

Adds a new PivotTable report. Returns a [PivotTable](#) object.

*expression*.**Add(PivotCache, TableDestination, TableName, ReadData, DefaultVersion)**

*expression* Required. An expression that returns a **PivotTables** object.

**PivotCache** Required **PivotCache**. The PivotTable cache on which the new PivotTable report is based. The cache provides data for the report.

**TableDestination** Required **Variant**. The cell in the upper-left corner of the PivotTable report's destination range (the range on the worksheet where the resulting report will be placed). You must specify a destination range on the worksheet that contains the **PivotTables** object specified by *expression* .

**TableName** Optional **Variant**. The name of the new PivotTable report.

**ReadData** Optional **Variant**. **True** to create a PivotTable cache that contains all records from the external database; this cache can be very large. **False** to enable setting some of the fields as server-based page fields before the data is actually read.

**DefaultVersion** Optional **Variant**. The version of Microsoft Excel the PivotTable was originally created in.



[Add method as it applies to the \*\*PublishObjects\*\* object.](#)

Creates an object that represents an item in a document saved to a Web page. Such objects facilitate subsequent updates to the Web page while automated changes are being made to the document in Microsoft Excel. Returns a **PublishObject** object.

*expression*.**Add(SourceType, FileName, Sheet, Source, HtmlType, DivID, Title)**

*expression* Required. An expression that returns a **PublishObjects** object.

**SourceType** Required **XlSourceType**. The source type.

XlSourceType can be one of these XlSourceType constants. Identifies the source object.

**xlSourceAutoFilter** An AutoFilter range.

**xlSourceChart** A chart.

**xlSourcePivotTable** A PivotTable report.

**xlSourcePrintArea** A range of cells selected for printing.

**xlSourceQuery** A query table (external data range).

**xlSourceRange** A range of cells.

**xlSourceSheet** An entire worksheet.

**xlSourceWorkbook** A workbook.

**FileName** Required **String**. The URL (on the intranet or the Web) or path (local or network) to which the source object was saved.

**Sheet** Optional **Variant**. The name of the worksheet that was saved as a Web page.

**Source** Optional **Variant**. A unique name used to identify items that have one of the following constants as their **SourceType** argument: **xlSourceAutoFilter**, **xlSourceChart**, **xlSourcePivotTable**, **xlSourcePrintArea**, **xlSourceQuery**, or **xlSourceRange**. If **SourceType** is **xlSourceRange**, **Source** specifies a range, which can be a defined name. If **SourceType** is **xlSourceChart**, **xlSourcePivotTable**, or **xlSourceQuery**, **Source** specifies the name of a chart, PivotTable report, or query table.

**HtmlType** Optional **Variant**. Specifies whether the item is saved as an interactive Microsoft Office Web component or as static text and images. Can be one of the **XIHTMLType** constants listed in the following table.

Constant	Description
<b>xlSourceAutoFilter</b>	An AutoFilter range
<b>xlSourceChart</b>	A chart
<b>xlSourcePivotTable</b>	A PivotTable report
<b>xlSourcePrintArea</b>	A range of cells selected for printing
<b>xlSourceQuery</b>	A query table (external data range)
<b>xlSourceRange</b>	A range of cells
<b>xlSourceSheet</b>	An entire worksheet

**DivID** Optional **Variant**. The unique identifier used in the HTML DIV tag to identify the item on the Web page.

**Title** Optional **Variant**. The title of the Web page.

[Add method as it applies to the \*\*QueryTables\*\* object.](#)

Creates a new query table. Returns a **QueryTable** object that represents the new query table.

## *expression*.Add(**Connection**, **Destination**, **Sql**)

*expression* Required. An expression that returns a **QueryTables** object.

**Connection** Required **VARIANT**. The data source for the query table. Can be one of the following:

- A string containing an OLE DB or ODBC connection string. The ODBC connection string has the form "ODBC;<connection string>".
- A **QueryTable** object from which the query information is initially copied, including the connection string and the SQL text, but not including the **Destination** range. Specifying a **QueryTable** object causes the **Sql** argument to be ignored.
- An ADO or DAO **Recordset** object. Data is read from the ADO or DAO recordset. Microsoft Excel retains the recordset until the query table is deleted or the connection is changed. The resulting query table cannot be edited.
- A Web query. A string in the form "URL;<url>", where "URL;" is required but not localized and the rest of the string is used for the URL of the Web query.
- Data Finder. A string in the form "FINDER;<data finder file path>" where "FINDER;" is required but not localized. The rest of the string is the path and file name of a Data Finder file (\*.dqy or \*.iqy). The file is read when the **Add** method is run; subsequent calls to the **Connection** property of the query table will return strings beginning with "ODBC;" or "URL;" as appropriate.
- A text file. A string in the form "TEXT;<text file path and name>", where TEXT is required but not localized.

**Destination** Required **RANGE**. The cell in the upper-left corner of the query table destination range (the range where the resulting query table will be placed). The destination range must be on the worksheet that contains the **QueryTables** object specified by *expression* .

**Sql** Optional **VARIANT**. The SQL query string to be run on the ODBC data source. This argument is optional when you're using an ODBC data source (if you don't specify it here, you should set it by using the **Sql** property of the query table before the table is refreshed). You cannot use this argument when a **QueryTable** object, text file, or ADO or DAO **Recordset** object is specified as

the data source.

## Remarks

A query created by this method isn't run until the [Refresh](#) method is called.

[Add method as it applies to the \*\*RecentFiles\*\* object.](#)

Adds a file to the list of recently used files. Returns a [RecentFile](#) object.

*expression.Add(Name)*

*expression* Required. An expression that returns a **RecentFiles** object.

**Name** Required **String**. The file name.

[Add method as it applies to the \*\*Scenarios\*\* object.](#)

Creates a new scenario and adds it to the list of scenarios that are available for the current worksheet. Returns a [Scenario](#) object.

*expression.Add(Name, ChangingCells, Values, Comment, Locked, Hidden)*

*expression* Required. An expression that returns a **Scenarios** object.

**Name** Required **String**. The scenario name.

**ChangingCells** Required **Variant**. A **Range** object that refers to the changing cells for the scenario.

**Values** Optional **Variant**. An array that contains the scenario values for the cells in **ChangingCells**. If this argument is omitted, the scenario values are assumed to be the current values in the cells in **ChangingCells**.

**Comment** Optional **Variant**. A string that specifies comment text for the scenario. When a new scenario is added, the author's name and date are automatically added at the beginning of the comment text.

**Locked** Optional **Variant**. **True** to lock the scenario to prevent changes. The default value is **True**.

***Hidden*** Optional **Variant. True** to hide the scenario. The default value is **False**.

## Remarks

A scenario name must be unique; Microsoft Excel generates an error if you try to create a scenario with a name that's already in use.

[Add method as it applies to the \*\*SeriesCollection\*\* object.](#)

Adds one or more new series to the **SeriesCollection** collection.

*expression*.**Add(Source, Rowcol, SeriesLabels, CategoryLabels, Replace)**

*expression* Required. An expression that returns a **SeriesCollection** object.

**Source** Required **Variant**. The new data, either as a **Range** object or an array of data points.

**Rowcol** Optional **XlRowCol**. Specifies whether the new values are in the rows or columns of the specified range.

XlRowCol can be one of these XlRowCol constants.

**xlColumns** *default*

**xlRows**

**SeriesLabels** Optional **Variant**. Ignored if **Source** is an array. **True** if the first row or column contains the name of the data series. **False** if the first row or column contains the first data point of the series. If this argument is omitted, Microsoft Excel attempts to determine the location of the series name from the contents of the first row or column.

**CategoryLabels** Optional **Variant**. Ignored if **Source** is an array. **True** if the first row or column contains the name of the category labels. **False** if the first row or column contains the first data point of the series. If this argument is omitted, Microsoft Excel attempts to determine the location of the category label from the contents of the first row or column.

**Replace** Optional **Variant**. If **CategoryLabels** is **True** and **Replace** is **True**, the specified categories replace the categories that currently exist for the series. If

***Replace*** is **False**, the existing categories will not be replaced. The default value is **False**.

## Remarks

This method does not actually return a **SeriesCollection** object as stated in the Object Browser. This method is not available for PivotChart reports.

[Add method as it applies to the \*\*Sheets\*\* and \*\*Worksheets\*\* objects.](#)

Creates a new worksheet, chart, or macro sheet. The new worksheet becomes the active sheet.

*expression*.**Add**(*Before*, *After*, *Count*, *Type*)

*expression* Required. An expression that returns one of the above objects.

**Before** Optional **Variant**. An object that specifies the sheet before which the new sheet is added.

**After** Optional **Variant**. An object that specifies the sheet after which the new sheet is added.

**Count** Optional **Variant**. The number of sheets to be added. The default value is one.

**Type** Optional **Variant**. Specifies the sheet type. Can be one of the following **XlSheetType** constants: **xlWorksheet**, **xlChart**, **xlExcel4MacroSheet**, or **xlExcel4IntlMacroSheet**. If you are inserting a sheet based on an existing template, specify the path to the template. The default value is **xlWorksheet**.

## Remarks

If **Before** and **After** are both omitted, the new sheet is inserted before the active sheet.

[Add method as it applies to the \*\*SmartTags\*\* object.](#)

Adds a smart tag. Returns a **SmartTag** object.

*expression*.**Add(SmartTagType)**

*expression* Required. An expression that returns a **SmartTags** object.

**SmartTagType** Required **String**. The type of smart tag.

[Add method as it applies to the \*\*Styles\*\* object.](#)

Creates a new style and adds it to the list of styles that are available for the current workbook. Returns a **Style** object.

*expression*.**Add(Name, BasedOn)**

*expression* Required. An expression that returns a **Styles** object.

**Name** Required **String**. The new style name.

**BasedOn** Optional **Variant**. A **Range** object that refers to a cell that's used as the basis for the new style. If this argument is omitted, the newly created style is based on the Normal style.

## Remarks

If a style with the specified name already exists, this method redefines the existing style based on the cell specified in **BasedOn**. The following example redefines the Normal style based on the active cell.

```
ActiveWorkbook.Styles.Add Name := "Normal", BasedOn := ActiveCell
```



[Add method as it applies to the Trendlines object.](#)

Creates a new trendline. Returns a [Trendline](#) object.

*expression.Add(Type, Order, Period, Forward, Backward, Intercept, DisplayEquation, DisplayRSquared, Name)*

*expression* Required. An expression that returns a **Trendlines** object.

**Type** Optional [XLTrendlineType](#). The trendline type.

XLTrendlineType can be one of these XLTrendlineType constants.

**xlExponential**

**xlLinear** *default*

**xlLogarithmic**

**xlMovingAvg**

**xlPolynomial**

**xlPower**

**Order** Optional **Variant**. Optional **Variant**. Required if **Type** is **xlPolynomial**. The trendline order. Must be an integer from 2 to 6, inclusive.

**Period** Optional **Variant**. Required if **Type** is **xlMovingAvg**. The trendline period. Must be an integer greater than 1 and less than the number of data points in the series you're adding a trendline to.

**Forward** Optional **Variant**. The number of periods (or units on a scatter chart) that the trendline extends forward.

**Backward** Optional **Variant**. The number of periods (or units on a scatter chart) that the trendline extends backward.

**Intercept** Optional **Variant**. The trendline intercept. If this argument is omitted, the intercept is automatically set by the regression.

**DisplayEquation** Optional **Variant**. **True** to display the equation of the trendline on the chart (in the same data label as the R-squared value). The default value is **False**.

**DisplayRSquared** Optional **Variant**. **True** to display the R-squared value of the trendline on the chart (in the same data label as the equation). The default value is **False**.

**Name** Optional **Variant**. The name of the trendline as text. If this argument is omitted, Microsoft Excel generates a name.



[Add method as it applies to the \*\*UserAccessList\*\* object.](#)

Adds a user access list. Returns a **UserAccess** object.

*expression*.**Add**(*Name*, *AllowEdit*)

*expression* Required. An expression that returns a **UserAccessList** object.

**Name** Required **String**. The name of the user access list.

**AllowEdit** Required **Boolean**. **True** allows users on the access list to edit the editable ranges on a protected worksheet.



[Add method as it applies to the \*\*Validation\*\* object.](#)

Adds data validation to the specified range.

*expression*.**Add**(*Type*, *AlertStyle*, *Operator*, *Formula1*, *Formula2*)

*expression* Required. An expression that returns a **Validation** object.

**Type** Required **XIDVType**. The validation type.

XIDVType can be one of these XIDVType constants.

**xlValidateCustom**

**xlValidateDate**

**xlValidateDecimal**

**xlValidateInputOnly**

**xlValidateList**

**xlValidateTextLength**

**xlValidateTime**

**xlValidateWholeNumber**

**AlertStyle** Optional **Variant**. The validation alert style. Can be one of the following **XIDVAlertStyle** constants: **xlValidAlertInformation**, **xlValidAlertStop**, or **xlValidAlertWarning**.

**Operator** Optional **Variant**. The data validation operator. Can be one of the following **XIFormatConditionOperator** constants: **xlBetween**, **xlEqual**, **xlGreater**, **xlGreaterEqual**, **xlLess**, **xlLessEqual**, **xlNotBetween**, or **xlNotEqual**.

**Formula1** Optional **Variant**. The first part of the data validation equation.

**Formula2** Optional **Variant**. The second part of the data validation when **Operator** is **xlBetween** or **xlNotBetween** (otherwise, this argument is ignored).

## Remarks

The **Add** method requires different arguments, depending on the validation type, as shown in the following table.

Validation type	Arguments
<b>xlValidateCustom</b>	<i>Formula1</i> is required, <i>Formula2</i> is ignored. <i>Formula1</i> must contain an expression that evaluates to <b>True</b> when data entry is valid and <b>False</b> when data entry is invalid.
<b>xlInputOnly</b>	<i>AlertStyle</i> , <i>Formula1</i> , or <i>Formula2</i> are used. <i>Formula1</i> is required, <i>Formula2</i> is ignored.
<b>xlValidateList</b>	<i>Formula1</i> must contain either a comma-delimited list of values or a worksheet reference to this list.
<b>xlValidateWholeNumber,</b> <b>xlValidateDate,</b> <b>xlValidateDecimal,</b> <b>xlValidateTextLength,</b> or <b>xlValidateTime</b>	One of either <i>Formula1</i> or <i>Formula2</i> must be specified, or both may be specified.

[Add method as it applies to the \*\*VPageBreaks\*\* object.](#)

Adds a vertical page break. Returns a **VPageBreak** object.

*expression*.**Add(Before)**

*expression* Required. An expression that returns a **VPageBreaks** object.

**Before** Required **Object**. A **Range** object. The range to the left of which the new page break will be added.

[Add method as it applies to the \*\*Watches\*\* object.](#)

Adds a range which is tracked when the worksheet is recalculated. Returns a **Watch** object.

*expression*.**Add(Source)**

*expression* Required. An expression that returns a **Watches** object.

**Source** Required **Variant**. The source for the range.



[Add method as it applies to the \*\*Workbooks\*\* object.](#)

Creates a new workbook. The new workbook becomes the active workbook.  
Returns a **Workbook** object.

*expression*.**Add(Template)**

*expression* Required. An expression that returns a **Workbooks** object.

**Template** Optional **Variant**. Determines how the new workbook is created. If this argument is a string specifying the name of an existing Microsoft Excel file, the new workbook is created with the specified file as a template. If this argument is a constant, the new workbook contains a single sheet of the specified type. Can be one of the following **XIWBATemplate** constants: **xlWBATChart**, **xlWBATExcel4IntlMacroSheet**, **xlWBATExcel4MacroSheet**, or **xlWBATWorksheet**. If this argument is omitted, Microsoft Excel creates a new workbook with a number of blank sheets (the number of sheets is set by the **SheetsInNewWorkbook** property).

## Remarks

If the *Template* argument specifies a file, the file name can include a path.



[Add method as it applies to the \*\*XmlMaps\*\* collection.](#)

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Adds an XML map to the specified workbook. Returns an **XmlMap** object.

*expression*.**Add**(*Schema*, *RootElementName*)

*expression* Required. An expression that returns a **Workbook** object.

**Schema** Required **String**. The schema to be added as an XML map. The string can be a path to a schema file, or the schema itself. The path can be specified in the Universal Naming Convention (UNC) or Uniform Resource Locator (URL) format.

**RootElementName** Optional **String**. The name of the root element. This argument can be ignored if the schema contains only one root element.

## Example

[As it applies to the \*\*AddIns\*\* object.](#)

This example inserts the add-in Myaddin.xla from drive A. When you run this example, Microsoft Excel copies the file A:\Myaddin.xla to the Library folder on your hard disk and adds the add-in title to the list in the **Add-Ins** dialog box.

```
Sub UseAddIn()  
  
    Set myAddIn = AddIns.Add(Filename:="A:\MYADDIN.XLA", _  
        CopyFile:=True)  
    MsgBox myAddIn.Title & " has been added to the list"  
  
End Sub
```

[As it applies to the \*\*AllowEditRanges\*\* object.](#)

This example allows edits to range "A1:A4" on the active worksheet, notifies the user, then changes the password for this specified range and notifies the user of this change.

```
Sub UseChangePassword()  
  
    Dim wksOne As Worksheet  
  
    Set wksOne = Application.ActiveSheet  
  
    ' Protect the worksheet.  
    wksOne.Protect  
  
    ' Establish a range that can allow edits  
    ' on the protected worksheet.  
    wksOne.Protection.AllowEditRanges.Add _  
        Title:="Classified", _  
        Range:=Range("A1:A4"), _  
        Password:="secret"  
  
    MsgBox "Cells A1 to A4 can be edited on the protected worksheet."  
  
    ' Change the password.  
    wksOne.Protection.AllowEditRanges(1).ChangePassword _
```

```

        Password:="moresecret"

        MsgBox "The password for these cells has been changed."

End Sub

```

 [As it applies to the \*\*CalculatedFields\*\* object.](#)

This example adds a calculated field to the first PivotTable report on worksheet one.

```

Worksheets(1).PivotTables(1).CalculatedFields.Add "PxS", _
    "= Product * Sales"

```

 [As it applies to the \*\*CalculatedMembers\*\* object.](#)

The following example adds a set to a PivotTable, assuming a PivotTable exists on the active worksheet.

```

Sub UseAddSet()

    Dim pvtOne As PivotTable
    Dim strAdd As String
    Dim strFormula As String
    Dim cbfOne As CubeField

    Set pvtOne = ActiveSheet.PivotTables(1)

    strAdd = "[MySet]"
    strFormula = "'{[Product].[All Products].[Food].children}'"

    ' Establish connection with data source if necessary.
    If Not pvtOne.PivotCache.IsConnected Then pvtOne.PivotCache.Make

    ' Add a calculated member titled "[MySet]"
    pvtOne.CalculatedMembers.Add Name:=strAdd, _
        Formula:=strFormula, Type:=xlCalculatedSet

    ' Add a set to the CubeField object.
    Set cbfOne = pvtOne.CubeFields.AddSet(Name:="[MySet]", _
        Caption:="My Set")

End Sub

```

 [As it applies to the \*\*ChartObjects\*\* object.](#)

This example creates a new embedded chart..

```
Set co = Sheets("Sheet1").ChartObjects.Add(50, 40, 200, 100)
co.Chart.ChartWizard Source:=Worksheets("Sheet1").Range("A1:B2"), _
    Gallery:=xlColumn, Format:=6, PlotBy:=xlColumns, _
    CategoryLabels:=1, SeriesLabels:=0, HasLegend:=1
```

 [As it applies to the \*\*Charts\*\* object.](#)

This example creates an empty chart sheet and inserts it before the last worksheet.

```
ActiveWorkbook.Charts.Add Before:=Worksheets(Worksheets.Count)
```

 [As it applies to the \*\*CustomProperties\*\* object.](#)

This example adds identifier information to the active worksheet and returns the name and value to the user.

```
Sub CheckCustomProperties()
    Dim wksSheet1 As Worksheet
    Set wksSheet1 = Application.ActiveSheet
    ' Add metadata to worksheet.
    wksSheet1.CustomProperties.Add _
        Name:="Market", Value:="Nasdaq"
    ' Display metadata.
    With wksSheet1.CustomProperties.Item(1)
        MsgBox .Name & vbTab & .Value
    End With
End Sub
```

 [As it applies to the \*\*CustomViews\*\* object.](#)

This example creates a new custom view named "Summary" in the active

workbook.

```
ActiveWorkbook.CustomViews.Add "Summary", True, True
```

[As it applies to the \*\*FormatConditions\*\* object.](#)

This example adds a conditional format to cells E1:E10.

```
With Worksheets(1).Range("e1:e10").FormatConditions _  
    .Add(xlCellValue, xlGreater, "=$a$1")  
    With .Borders  
        .LineStyle = xlContinuous  
        .Weight = xlThin  
        .ColorIndex = 6  
    End With  
    With .Font  
        .Bold = True  
        .ColorIndex = 3  
    End With  
End With
```

[As it applies to the \*\*HPageBreaks\*\* object.](#)

This example adds a horizontal page break above cell F25 and adds a vertical page break to the left of this cell.

```
With Worksheets(1)  
    .HPageBreaks.Add .Range("F25")  
    .VPageBreaks.Add .Range("F25")  
End With
```

[As it applies to the \*\*Hyperlinks\*\* object.](#)

This example adds a hyperlink to cell A5.

```
With Worksheets(1)  
    .Hyperlinks.Add Anchor:=.Range("a5"), _  
        Address:="http://example.microsoft.com", _  
        ScreenTip:="Microsoft Web Site", _  
        TextToDisplay:="Microsoft"  
End With
```

This example adds an email hyperlink to cell A5.

```
With Worksheets(1)
    .Hyperlinks.Add Anchor:=.Range("a5"), _
        Address:="mailto:someone@microsoft.com?subject=hello", _
        ScreenTip:="Write us today", _
        TextToDisplay:="Support"
End With
```

[As it applies to the \*\*ListColumns\*\* collection object.](#)

The following example adds a new column to the default **ListObject** object in the first worksheet of the workbook. Because no position is specified, a new rightmost column is added.

```
Set myNewColumn = ActiveWorkbook.Worksheets(1).ListObjects(1).ListCo
```

**Note** A name for the column is automatically generated. You can choose to change the name after the column has been added.

[As it applies to the \*\*ListObjects\*\* collection object.](#)

The following example adds a new **ListObject** object based on data from a Microsoft Windows SharePoint Services site to the default **ListObjects** collection and places the list in cell A1 in the first worksheet of the workbook.

**Note** The following code example assumes that you will substitute a valid server name and the list guid in the variables **strServerName** and **strListGUID**. Additionally, the server name must be followed by `"/_vti_bin"` or the sample will not work.

```
Set objListObject = ActiveWorkbook.Worksheets(1).ListObjects.Add(Sou
    Source:= Array(strServerName, StrListGUID), TRUE, xlGuess, Des
```

**Note** If there is existing data at cell A1, the existing list data will be moved to the right to accommodate the new list.

[As it applies to the \*\*ListRows\*\* collection object.](#)

The following example adds a new row to the default **ListObject** object in the

first worksheet of the workbook. Because no position is specified, the new row is added to the bottom of the list.

```
Set myNewColumn = ActiveWorkbook.Worksheets(1).ListObject(1).ListRow
```



[As it applies to the \*\*Names\*\* object.](#)

This example defines a new name for the range A1:D3 on Sheet1 in the active workbook. **Note** Nothing is returned if Sheet1 does not exist.

```
Sub MakeRange()  
  
    ActiveWorkbook.Names.Add _  
        Name:="tempRange", _  
        RefersTo:="=Sheet1!$A$1:$D$3"  
  
End Sub
```



[As it applies to the \*\*OLEObjects\*\* object.](#)

This example creates a new Microsoft Word OLE object on Sheet1.

```
ActiveWorkbook.Worksheets("Sheet1").OLEObjects.Add _  
    ClassType:="Word.Document"
```

This example adds a command button to sheet one.

```
Worksheets(1).OLEObjects.Add ClassType:="Forms.CommandButton.1", _  
    Link:=False, DisplayAsIcon:=False, Left:=40, Top:=40, _  
    Width:=150, Height:=10
```



[As it applies to the \*\*Parameters\*\* object.](#)

This example changes the SQL statement for query table one. The clause "(city=?)" indicates that the query is a parameter query, and the value of city is set to the constant "Oakland."

```
Set qt = Sheets("sheet1").QueryTables(1)  
qt.Sql = "SELECT * FROM authors WHERE (city=?)"  
Set param1 = qt.Parameters.Add("City Parameter", _  
    xlParamTypeVarChar)
```

```
param1.SetParam xlConstant, "Oakland"  
qt.Refresh
```

 [As it applies to the \*\*Phonetics\*\* object.](#)

This example adds three phonetic text strings to the active cell. The example then sets the character type to Hiragana, sets the font color to blue, and sets the text to visible.

```
ActiveCell.FormulaR1C1 = "東京都渋谷区代々木"  
ActiveCell.Phonetics.Add Start:=1, Length:=3, Text:="トウキョウト"  
ActiveCell.Phonetics.Add Start:=4, Length:=3, Text:="シブヤク"  
ActiveCell.Phonetics.CharacterType = xlHiragana  
ActiveCell.Phonetics.Font.Color = vbBlue  
ActiveCell.Phonetics.Visible = True
```

 [As it applies to the \*\*PivotCaches\*\* object.](#)

This example creates a new PivotTable cache based on an OLAP provider and then it creates a new PivotTable report based on the cache, at cell A3 on the active worksheet.

```
Dim cnnConn As ADODB.Connection  
Dim rstRecordset As ADODB.Recordset  
Dim cmdCommand As ADODB.Command  
  
' Open the connection.  
Set cnnConn = New ADODB.Connection  
With cnnConn  
    .ConnectionString = _  
        "Provider=Microsoft.Jet.OLEDB.4.0"  
    .Open "C:\perfdate\record.mdb"  
End With  
  
' Set the command text.  
Set cmdCommand = New ADODB.Command  
Set cmdCommand.ActiveConnection = cnnConn  
With cmdCommand  
    .CommandText = "Select Speed, Pressure, Time From DynoRun"  
    .CommandType = adCmdText  
    .Execute  
End With
```

```

' Open the recordset.
Set rstRecordset = New ADODB.Recordset
Set rstRecordset.ActiveConnection = cnnConn
rstRecordset.Open cmdCommand

' Create a PivotTable cache and report.
Set objPivotCache = ActiveWorkbook.PivotCaches.Add( _
    SourceType:=xlExternal)
Set objPivotCache.Recordset = rstRecordset
With objPivotCache
    .CreatePivotTable TableDestination:=Range("A3"), _
        TableName:="Performance"
End With

With ActiveSheet.PivotTables("Performance")
    .SmallGrid = False
    With .PivotFields("Pressure")
        .Orientation = xlRowField
        .Position = 1
    End With
    With .PivotFields("Speed")
        .Orientation = xlColumnField
        .Position = 1
    End With
    With .PivotFields("Time")
        .Orientation = xlDataField
        .Position = 1
    End With
End With

' Close the connections and clean up.
cnnConn.Close
Set cmdCommand = Nothing
Set rstRecordSet = Nothing
Set cnnConn = Nothing

```

[As it applies to the \*\*PivotFormulas\*\* object.](#)

This example creates a new PivotTable formula for the first PivotTable report on worksheet one.

```

Worksheets(1).PivotTables(1).PivotFormulas _
    .Add "Year['1998'] Apples = (Year['1997'] Apples) * 2"

```

[As it applies to the \*\*PivotItems\*\* object.](#)

This example creates a new PivotTable item in the first PivotTable report on worksheet one.

```
Worksheets(1).PivotTables(1).PivotItems("Year").Add "1998"
```



[As it applies to the \*\*PivotTables\*\* object.](#)

This example creates a new PivotTable cache based on an [OLAP provider](#), and then it creates a new PivotTable report based on the cache, at cell A1 on the first worksheet.

```
Dim cnnConn As ADODB.Connection
Dim rstRecordset As ADODB.Recordset
Dim cmdCommand As ADODB.Command

' Open the connection.
Set cnnConn = New ADODB.Connection
With cnnConn
    .ConnectionString = _
        "Provider=Microsoft.Jet.OLEDB.4.0"
    .Open "C:\perfdate\record.mdb"
End With

' Set the command text.
Set cmdCommand = New ADODB.Command
Set cmdCommand.ActiveConnection = cnnConn
With cmdCommand
    .CommandText = "Select Speed, Pressure, Time From DynoRun"
    .CommandType = adCmdText
    .Execute
End With

' Open the recordset.
Set rstRecordset = New ADODB.Recordset
Set rstRecordset.ActiveConnection = cnnConn
rstRecordset.Open cmdCommand

' Create PivotTable cache and report.
Set objPivotCache = ActiveWorkbook.PivotCaches.Add( _
    SourceType:=xlExternal)
Set objPivotCache.Recordset = rstRecordset

ActiveSheet.PivotTables.Add _
    PivotCache:=objPivotCache, _
    TableDestination:=Range("A3"), _
    TableName:="Performance"
```

```

With ActiveSheet.PivotTables("Performance")
    .SmallGrid = False
    With .PivotFields("Pressure")
        .Orientation = xlRowField
        .Position = 1
    End With
    With .PivotFields("Speed")
        .Orientation = xlColumnField
        .Position = 1
    End With
    With .PivotFields("Time")
        .Orientation = xlDataField
        .Position = 1
    End With
End With

' Close the connections and clean up.
cnnConn.Close
Set cmdCommand = Nothing
Set rstRecordSet = Nothing
Set cnnConn = Nothing

```

 [As it applies to the \*\*PublishObjects\*\* object.](#)

This example saves the range D5:D9 on the First Quarter worksheet in the active workbook to a Web page called Stockreport.htm. You use the Spreadsheet component to add interactivity to the Web page.

```

ActiveWorkbook.PublishObjects.Add( _
    SourceType:=xlSourceRange, _
    Filename:="\\Server2\Q1\Stockreport.htm", _
    Sheet:"First Quarter", _
    Source:"D5:D9", _
    HTMLType:=xlHTMLCalc).Publish

```

 [As it applies to the \*\*QueryTables\*\* object.](#)

This example creates a query table based on an ADO recordset. The example preserves the existing column sorting and filtering settings and layout information for backward compatibility.

```

Dim cnnConnect As ADODB.Connection
Dim rstRecordset As ADODB.Recordset

```

```

Set cnnConnect = New ADODB.Connection
cnnConnect.Open "Provider=SQLOLEDB;" & _
    "Data Source=srvdata;" & _
    "User ID=testac;Password=4me2no;"

Set rstRecordset = New ADODB.Recordset
rstRecordset.Open _
    Source:="Select Name, Quantity, Price From Products", _
    ActiveConnection:=cnnConnect, _
    CursorType:=adOpenDynamic, _
    LockType:=adLockReadOnly, _
    Options:=adCmdText

With ActiveSheet.QueryTables.Add( _
    Connection:=rstRecordset, _
    Destination:=Range("A1"))
    .Name = "Contact List"
    .FieldNames = True
    .RowNumbers = False
    .FillAdjacentFormulas = False
    .PreserveFormatting = True
    .RefreshOnFileOpen = False
    .BackgroundQuery = True
    .RefreshStyle = xlInsertDeleteCells
    .SavePassword = True
    .SaveData = True
    .AdjustColumnWidth = True
    .RefreshPeriod = 0
    .PreserveColumnInfo = True
    .Refresh BackgroundQuery:=False
End With

```

This example imports a fixed width text file into a new query table. The first column in the text file is five characters wide and is imported as text. The second column is four characters wide and is skipped. The remainder of the text file is imported into the third column and has the General format applied to it.

```

Set shFirstQtr = Workbooks(1).Worksheets(1)
Set qtQtrResults = shFirstQtr.QueryTables.Add( _
    Connection := "TEXT;C:\My Documents\19980331.txt",
    Destination := shFirstQtr.Cells(1,1))
With qtQtrResults
    .TextFileParsingType = xlFixedWidth
    .TextFileFixedColumnWidths := Array(5,4)
    .TextFileColumnDataTypes := _
        Array(xlTextFormat, xlSkipColumn, xlGeneralFormat)

```

```
.Refresh  
End With
```

This example creates a new query table on the active worksheet.

```
sqlstring = "select 96Sales.totals from 96Sales where profit < 5"  
connstring = _  
    "ODBC;DSN=96SalesData;UID=Rep21;PWD=NUyHwYQI;Database=96Sales"  
With ActiveSheet.QueryTables.Add(Connection:=connstring, _  
    Destination:=Range("B1"), Sql:=sqlstring)  
    .Refresh  
End With
```

[As it applies to the \*\*RecentFiles\*\* object.](#)

This example adds Oscar.xls to the list of recently used files.

```
Application.RecentFiles.Add Name:="Oscar.xls"
```

[As it applies to the \*\*Scenarios\*\* object.](#)

This example adds a new scenario to Sheet1.

```
Worksheets("Sheet1").Scenarios.Add Name:="Best Case", _  
    ChangingCells:=Worksheets("Sheet1").Range("A1:A4"), _  
    Values:=Array(23, 5, 6, 21), _  
    Comment:="Most favorable outcome."
```

[As it applies to the \*\*SeriesCollection\*\* object.](#)

This example creates a new series in Chart1. The data source for the new series is range B1:B10 on Sheet1.

```
Charts("Chart1").SeriesCollection.Add _  
    Source:=ActiveWorkbook.Worksheets("Sheet1").Range("B1:B10")
```

This example creates a new series on the embedded chart on Sheet1.

```
Worksheets("Sheet1").ChartObjects(1).Activate
```

```
ActiveChart.SeriesCollection.Add _  
    Source:=Worksheets("Sheet1").Range("B1:B10")
```

 [As it applies to the \*\*Sheets\*\* and \*\*WorkSheets\*\* objects.](#)

This example inserts a new worksheet before the last worksheet in the active workbook.

```
ActiveWorkbook.Sheets.Add Before:=Worksheets(Worksheets.Count)
```

 [As it applies to the \*\*SmartTags\*\* object.](#)

This example adds a smart tag titled MSFT to cell A1, then adds extra metadata called Market with the value of Nasdaq to the smart tag and then returns the value of the property to the user. This example assumes the host system is connected to the Internet.

```
Sub UseProperties()  
  
    Dim strLink As String  
    Dim strType As String  
  
    ' Define smart tag variables.  
    strLink = "urn:schemas-microsoft-com:smarctags#stocktickerSymbol  
    strType = "stockview"  
  
    Range("A1").Formula = "MSFT"  
  
    ' Add a property for MSFT smart tag and define its value.  
    Range("A1").SmartTags.Add(strLink).Properties.Add _  
        Name:="Market", Value:="Nasdaq"  
  
    ' Notify the user of the smart tag's value.  
    MsgBox Range("A1").SmartTags.Add(strLink).Properties("Market").v  
  
End Sub
```

 [As it applies to the \*\*Styles\*\* object.](#)

This example defines a new style based on cell A1 on Sheet1.

```

With ActiveWorkbook.Styles.Add(Name:="theNewStyle")
    .IncludeNumber = False
    .IncludeFont = True
    .IncludeAlignment = False
    .IncludeBorder = False
    .IncludePatterns = False
    .IncludeProtection = False
    .Font.Name = "Arial"
    .Font.Size = 18
End With

```

 [As it applies to the \*\*Trendlines\*\* object.](#)

This example creates a new linear trendline in Chart1.

```

ActiveWorkbook.Charts("Chart1").SeriesCollection(1).Trendlines.Add

```

 [As it applies to the \*\*Validation\*\* object.](#)

This example adds data validation to cell E5.

```

With Range("e5").Validation
    .Add Type:=xlValidateWholeNumber, _
        AlertStyle:= xlValidAlertStop, _
        Operator:=xlBetween, Formula1:="5", Formula2:="10"
    .InputTitle = "Integers"
    .ErrorTitle = "Integers"
    .InputMessage = "Enter an integer from five to ten"
    .ErrorMessage = "You must enter a number from five to ten"
End With

```

 [As it applies to the \*\*VPageBreaks\*\* object.](#)

This example adds a horizontal page break above cell F25 and adds a vertical page break to the left of this cell.

```

With Worksheets(1)
    .HPageBreaks.Add .Range("F25")
    .VPageBreaks.Add .Range("F25")
End With

```

[As it applies to the \*\*Watches\*\* object.](#)

This example creates a summation formula in cell A3 and then adds this cell to the watch facility.

```
Sub AddWatch()  
  
    With Application  
        .Range("A1").Formula = 1  
        .Range("A2").Formula = 2  
        .Range("A3").Formula = "=Sum(A1:A2)"  
        .Range("A3").Select  
        .Watches.Add Source:=ActiveCell  
    End With  
  
End Sub
```

[As it applies to the \*\*WorkBooks\*\* object.](#)

This example creates a new workbook.

```
workbooks.Add
```



[Show All](#)

# AddCallout Method

Creates a borderless line callout. Returns a [Shape](#) object that represents the new callout.

*expression*.**AddCallout**(*Type*, *Left*, *Top*, *Width*, *Height*)

*expression* Required. An expression that returns one of the objects in the Applies to List.

**Type** Required [MsoCalloutType](#). The type of callout line.

MsoCalloutType can be one of these MsoCalloutType constants.

**msoCalloutOne**. A single-segment callout line that can be either horizontal or vertical.

**msoCalloutTwo**. A single-segment callout line that rotates freely.

**msoCalloutMixed**.

**msoCalloutThree**. A two-segment line.

**msoCalloutFour**. A three-segment line.

**Left** Required **Single**. The position (in points) of the upper-left corner of the callout's bounding box relative to the upper-left corner of the document.

**Top** Required **Single**. The position (in points) of the upper-left corner of the callout's bounding box relative to the upper-left corner of the document.

**Width** Required **Single**. The width of the callout's bounding box, in points.

**Height** Required **Single**. The height of the callout's bounding box, in points.

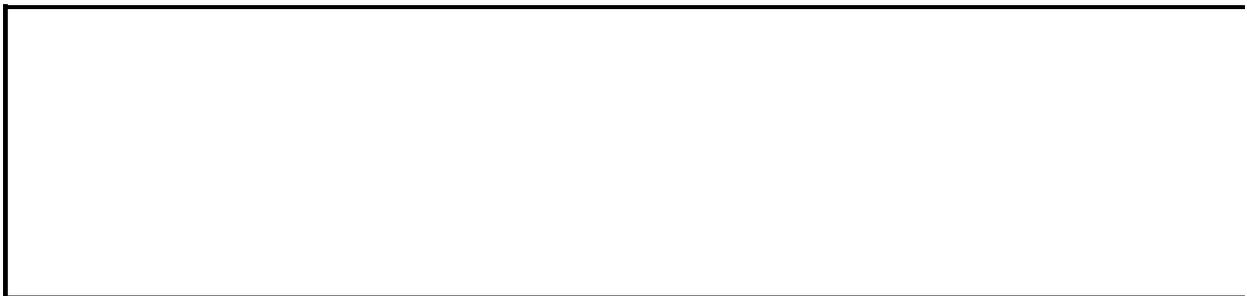
## Remarks

You can insert a greater variety of callouts by using the [AddShape](#) method.

## Example

This example adds a borderless callout with a freely rotating one-segment callout line to myDocument and then sets the callout angle to 30 degrees.

```
Set myDocument = Worksheets(1)
myDocument.Shapes.AddCallout(Type:=msoCalloutTwo, _
    Left:=50, Top:=50, Width:=200, Height:=100) _
    .Callout.Angle = msoCalloutAngle30
```



# AddChartAutoFormat Method

Adds a custom chart autoformat to the list of available chart autoformats.

*expression*.**AddChartAutoFormat**(*Chart*, *Name*, *Description*)

*expression* Required. An expression that returns an **Application** object.

**Chart** Required **Chart**. A chart that contains the format that will be applied when the new chart autoformat is applied.

**Name** Required **String**. The name of the autoformat.

**Description** Optional **String**. A description of the custom autoformat.

## Example

This example adds a new autofomat based on Chart1.

```
Application.AddChartAutoFormat _  
    Chart:=Charts("Chart1"), Name:="Presentation Chart"
```



# AddComment Method

Adds a comment to the range.

*expression*.**AddComment**(*Text*)

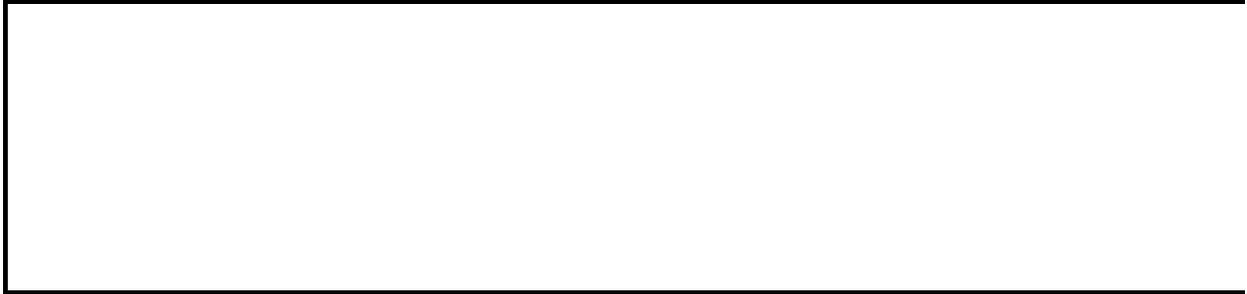
*expression* Required. An expression that returns a **Range** object.

*Text* Optional **Variant**. The comment text.

## Example

This example adds a comment to cell E5 on worksheet one.

```
Worksheets(1).Range("E5").AddComment "Current Sales"
```



[Show All](#)

# AddConnector Method

Creates a connector. Returns a [Shape](#) object that represents the new connector. When a connector is added, it's not connected to anything. Use the [BeginConnect](#) and [EndConnect](#) methods to attach the beginning and end of a connector to other shapes in the document.

*expression*.AddConnector(*Type*, *BeginX*, *BeginY*, *EndX*, *EndY*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*Type* Required [MsoConnectorType](#). The connector type to add.

MsoConnectorType can be one of these MsoConnectorType constants.

**msoConnectorElbow**

**msoConnectorTypeMixed**

**msoConnectorCurve**

**msoConnectorStraight**

**BeginX** Required **Single**. The horizontal position (in points) of the connector's starting point relative to the upper-left corner of the document.

**BeginY** Required **Single**. The vertical position (in points) of the connector's starting point relative to the upper-left corner of the document.

**EndX** Required **Single**. The horizontal position (in points) of the connector's end point relative to the upper-left corner of the document.

**EndY** Required **Single**. The vertical position (in points) of the connector's end point relative to the upper-left corner of the document.

## Remarks

When you attach a connector to a shape, the size and position of the connector are automatically adjusted, if necessary. Therefore, if you're going to attach a connector to other shapes, the position and dimensions you specify when adding the connector are irrelevant.

## Example

The following example adds a curved connector to a new canvas in a new worksheet.

```
Sub AddCanvasConnector()  
  
    Dim wksNew As Worksheet  
    Dim shpCanvas As Shape  
  
    Set wksNew = Worksheets.Add  
  
    'Add drawing canvas to new worksheet  
    Set shpCanvas = wksNew.Shapes.AddCanvas( _  
        Left:=150, Top:=150, Width:=200, Height:=300)  
  
    'Add connector to the drawing canvas  
    shpCanvas.CanvasItems.AddConnector _  
        Type:=msoConnectorStraight, BeginX:=150, _  
        BeginY:=150, EndX:=200, EndY:=200  
  
End Sub
```



[Show All](#)

# AddCurve Method

As it applies to the **Shapes** object, returns a [Shape](#) object that represents a Bézier curve in a worksheet. As it applies to the **CanvasShapes** object, returns a **Shape** object that represents a Bézier curve in a drawing canvas.

*expression.AddCurve(SafeArrayOfPoints)*

*expression* Required. An expression that returns one of the objects in the Applies To list.

**SafeArrayOfPoints** Required **Variant**. An array of [coordinate pairs](#) that specifies the vertices and control points of the curve. The first point you specify is the starting vertex, and the next two points are control points for the first Bézier segment. Then, for each additional segment of the curve, you specify a vertex and two control points. The last point you specify is the ending vertex for the curve. Note that you must always specify  $3n + 1$  points, where  $n$  is the number of segments in the curve.

## Example

The following example adds a two-segment Bézier curve to myDocument.

```
Dim pts(1 To 7, 1 To 2) As Single
pts(1, 1) = 0
pts(1, 2) = 0
pts(2, 1) = 72
pts(2, 2) = 72
pts(3, 1) = 100
pts(3, 2) = 40
pts(4, 1) = 20
pts(4, 2) = 50
pts(5, 1) = 90
pts(5, 2) = 120
pts(6, 1) = 60
pts(6, 2) = 30
pts(7, 1) = 150
pts(7, 2) = 90
Set myDocument = Worksheets(1)
myDocument.Shapes.AddCurve SafeArrayOfPoints:=pts
```



# AddCustomList Method

Adds a custom list for custom autofill and/or custom sort.

*expression*.AddCustomList(*ListArray*, *ByRow*)

*expression* Required. An expression that returns an **Application** object.

**ListArray** Required **Variant**. Specifies the source data, as either an array of strings or a **Range** object.

**ByRow** Optional **Variant**. Only used if **ListArray** is a **Range** object. **True** to create a custom list from each row in the range. **False** to create a custom list from each column in the range. If this argument is omitted and there are more rows than columns (or an equal number of rows and columns) in the range, Microsoft Excel creates a custom list from each column in the range. If this argument is omitted and there are more columns than rows in the range, Microsoft Excel creates a custom list from each row in the range.

## Remarks

If the list you're trying to add already exists, this method does nothing.

## Example

This example adds an array of strings as a custom list.

```
Application.AddCustomList Array("cogs", "sprockets", _  
    "widgets", "gizmos")
```



[Show All](#)

# AddDataField Method

Adds a data field to a PivotTable report. Returns a [PivotField](#) object that represents the new data field.

*expression*.**AddDataField**(*Field*, *Caption*, *Function*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Field** Required **Object**. The unique field on the server. If the source data is [Online Analytical Processing \(OLAP\)](#), the unique field is a cube field. If the source data is non-OLAP ([non-OLAP source data](#)), the unique field is a PivotTable field.

**Caption** Optional **VARIANT**. The label used in the PivotTable report to identify this data field.

**Function** Optional **VARIANT**. The function performed in the added data field.

## Example

This example adds a data field titled "Total Score" to a pivot table called "PivotTable1".

**Note** : This example assumes a table exists in which one of the columns contains a column titled "Score".

```
Sub AddMoreFields()  
    With ActiveSheet.PivotTables("PivotTable1")  
        .AddDataField ActiveSheet.PivotTables( _  
            "PivotTable1").PivotFields("Score"), "Total Score"  
    End With  
End Sub
```



[Show All](#)

# AddDiagram Method

Creates a diagram. Returns a [Shape](#) object that represents the new diagram.

*expression*.AddDiagram(*Type*, *Left*, *Top*, *Width*, *Height*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*Type* Required [MsoDiagramType](#). The type of diagram.

MsoDiagramType can be one of these MsoDiagramType constants.

**msoDiagramCycle** A process diagram with a continuous cycle diagram type.

**msoDiagramMixed** A mixed diagram type.

**msoDiagramOrgChart** A hierarchical relationship diagram type.

**msoDiagramPyramid** A foundation based relationships diagram type.

**msoDiagramRadial** A diagram type showing relationships of a core element.

**msoDiagramTarget** A diagram type showing steps toward a goal.

**msoDiagramVenn** A diagram type showing areas of overlap between elements.

*Left* Required **Single**. The position (in points) of the upper-left corner of the diagram relative to the upper-left corner of the worksheet.

*Top* Required **Single**. The position (in points) of the upper-left top of the diagram relative to the upper-left corner of the worksheet.

*Width* Required **Single**. The width of the diagram, in points.

*Height* Required **Single**. The height of the diagram, in points.

## Example

This example adds a pyramid diagram to the active sheet.

```
Sub CreatePyramidDiagram()  
  
    Dim dgnNode As DiagramNode  
    Dim shpDiagram As Shape  
    Dim intCount As Integer  
  
    'Add pyramid diagram to current document  
    Set shpDiagram = ActiveSheet.Shapes.AddDiagram _  
        (Type:=msoDiagramPyramid, Left:=10, _  
         Top:=15, Width:=400, Height:=475)  
    'Add first diagram node child to pyramid diagram  
    Set dgnNode = shpDiagram.DiagramNode.Children.AddNode  
  
    'Add three more diagram node children to the pyramid diagram  
    For intCount = 1 To 3  
        dgnNode.AddNode  
    Next intCount  
  
End Sub
```



[Show All](#)

# AddFields Method

Adds row, column, and page fields to a PivotTable report or PivotChart report.

*expression*.**AddFields**(*RowFields*, *ColumnFields*, *PageFields*, *AddToTable*, *AppendField*)

*expression* Required. An expression that returns a **PivotTable** object.

**RowFields** Optional **Variant**. Specifies a field name (or an array of field names) to be added as rows, or to be added to the category axis.

**ColumnFields** Optional **Variant**. Specifies a field name (or an array of field names) to be added as columns, or to be added to the series axis.

**PageFields** Optional **Variant**. Specifies a field name (or an array of field names) to be added as pages, or to be added to the page area.

**AddToTable** Optional **Variant**. Applies only to PivotTable reports. **True** to add the specified fields to the report (none of the existing fields are replaced). **False** to replace existing fields with the new fields. The default value is **False**.

**AppendField** Optional **Boolean**. Applies only to PivotChart reports. **True** to add the specified fields to the report (none of the existing fields are replaced). **False** to replace existing fields with the new fields. The default value is **False**.

## Remarks

You must specify one of the field arguments.

Field names specify the unique name returned by the **SourceName** property of the **PivotField** object.

This method is not available for [OLAP](#) data sources.

## Example

This example replaces the existing column fields in the first PivotTable report on Sheet1 with the Status and Closed\_By fields.

```
Worksheets("Sheet1").PivotTables(1).AddFields _  
    ColumnFields:=Array("Status", "Closed_By")
```



[Show All](#)

# AddFormControl Method

Creates a [Microsoft Excel control](#). Returns a **Shape** object that represents the new control.

*expression*.**AddFormControl**(*Type, Left, Top, Width, Height*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Type** Required [XlFormControl](#). The [Microsoft Excel control](#) type. You cannot create an edit box on a worksheet.

XlFormControl can be one of these XlFormControl constants.

**xlButtonControl**

**xlCheckBox**

**xlDropDown**

**xlEditBox**

**xlGroupBox**

**xlLabel**

**xlListBox**

**xlOptionButton**

**xlScrollBar**

**xlSpinner**

**Left** Required **Long**. The initial coordinates of the new object (in [points](#)) relative to the upper-left corner of cell A1 on a worksheet or to the upper-left corner of a chart.

**Top** Required **Long**. The initial coordinates of the new object (in [points](#)) relative to the upper-left corner of cell A1 on a worksheet or to the upper-left corner of a chart.

**Width** Required **Long**. The initial size of the new object, in points.

***Height*** Required **Long**. The initial size of the new object, in points.

## Remarks

Use the [AddOLEObject](#) method or the [Add](#) method of the **OLEObjects** collection to create an [ActiveX control](#).

## Example

This example adds a list box to worksheet one and sets the fill range for the list box.

```
With Worksheets(1)
    Set lb = .Shapes.AddFormControl(xlListBox, 100, 10, 100, 100)
    lb.ControlFormat.ListFillRange = "A1:A10"
End With
```



# AddItem Method

Adds an item to a list box or a combo box.

*expression*.**AddItem**(*Text*, *Index*)

*expression* Required. An expression that returns a **ControlFormat** object.

**Text** Required **String**. The text to be added

**Index** Optional **Variant**. The position of the new entry. If the list has fewer entries than the specified index, blank items from the end of the list are added to the specified position. If this argument is omitted, the item is appended to the existing list.

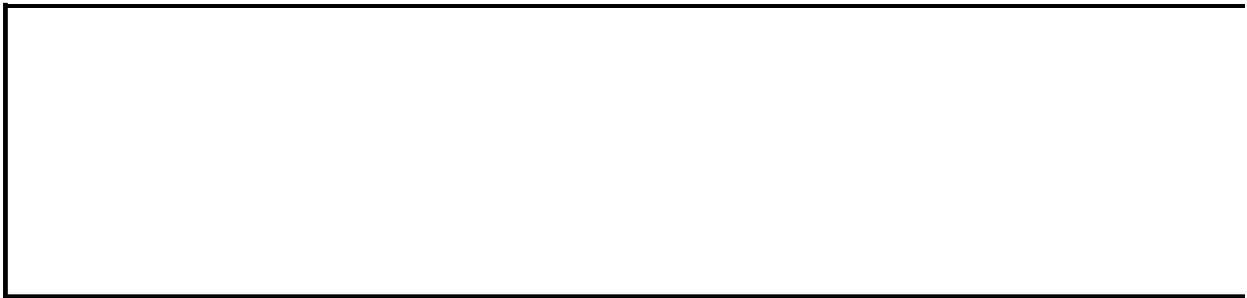
## Remarks

Using this method clears any range specified by the [ListFillRange](#) property.

## Example

This example creates a list box and fills it with integers from 1 to 10.

```
With Worksheets(1)
    Set lb = .Shapes.AddFormControl(xlListBox, 100, 10, 100, 100)
    For x = 1 To 10
        lb.ControlFormat.AddItem x
    Next
End With
```



[Show All](#)

# AddLabel Method

Creates a label. Returns a [Shape](#) object that represents the new label.

*expression*.AddLabel(*Orientation, Left, Top, Width, Height*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Orientation** Required [MsoTextOrientation](#). The text orientation within the label.

MsoTextOrientation can be one of these MsoTextOrientation constants.

**msoTextOrientationDownward**

**msoTextOrientationHorizontal**

**msoTextOrientationHorizontalRotatedFarEast**

**msoTextOrientationMixed**

**msoTextOrientationUpward**

**msoTextOrientationVertical**

**msoTextOrientationVerticalFarEast**

Some of these constants may not be available to you, depending on the language support (U.S. English, for example) that you've selected or installed.

**Left** Required **Single**. The position (in points) of the upper-left corner of the label relative to the upper-left corner of the document.

**Top** Required **Single**. The position (in points) of the upper-left corner of the label relative to the top corner of the document.

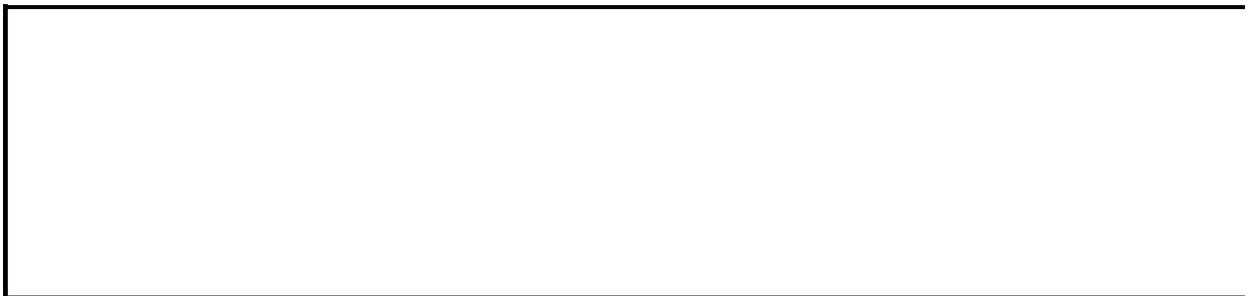
**Width** Required **Single**. The width of the label, in points.

**Height** Required **Single**. The height of the label, in points.

## Example

This example adds a vertical label that contains the text "Test Label" to myDocument.

```
Set myDocument = Worksheets(1)
myDocument.Shapes.AddLabel(msoTextOrientationVertical, _
    100, 100, 60, 150) _
    .TextFrame.Characters.Text = "Test Label"
```



# AddLine Method

As it applies to the **Shapes** object, returns a [Shape](#) object that represents the new line in a worksheet. As it applies to the **CanvasShapes** object, returns a **Shape** object that represents the new line in a drawing canvas.

*expression*.AddLine(*BeginX*, *BeginY*, *EndX*, *EndY*)

*expression* Required. An expression that returns a **Shapes** object.

**BeginX** , **BeginY** Required **Single**. The position (in points) of the line's starting point relative to the upper-left corner of the document.

**EndX** , **EndY** Required **Single**. The position (in points) of the line's end point relative to the upper-left corner of the document.

## Example

This example adds a blue dashed line to myDocument.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddLine(10, 10, 250, 250).Line
    .DashStyle = msoLineDashDotDot
    .ForeColor.RGB = RGB(50, 0, 128)
End With
```



# AddMemberPropertyField Method

Adds a member property field to the display for the cube field.

*expression*.AddMemberPropertyField(*Property*, *PropertyOrder*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Property** Required **String**. The unique name of the member property. For balanced hierarchies, a unique name can be created by appending the "quoted" member property name to the unique name of the level the member property is associated with. For unbalanced hierarchies, a unique name can be created by appending the "quoted" member property name to the unique name of the hierarchy.

**PropertyOrder** Optional **Variant**. Sets the **PropertyOrder** property value for a **CubeField** object. The actual position in the collection will be immediately before the PivotTable field that currently has the same **PropertyOrder** value that is given in the argument. If no field has the given property order value, the range of acceptable values is 1 to the number of member properties already showing for the hierarchy plus one. This argument is one-based. If omitted, the property goes to the end of the list.

## Remarks

The property field specified will not be viewable if the PivotTable view has no fields.

To delete member properties, use the **Delete** method to delete the **PivotField** object from the **PivotFields** collection.

## Example

In this example, Microsoft Excel adds a member property field titled "Description" to the PivotTable report view. This example assumes that a PivotTable exists on the active worksheet and that "Country", "Area" and "Description" are items in the report.

```
Sub UseAddMemberPropertyField()  
  
    Dim pvtTable As PivotTable  
  
    Set pvtTable = ActiveSheet.PivotTables(1)  
  
    With pvtTable  
        .ManualUpdate = True  
        .CubeFields("[Country]").LayoutForm = xlOutline  
        .CubeFields("[Country]").AddMemberPropertyField _  
            Property:="[Country].[Area].[Description]"  
        .ManualUpdate = False  
    End With  
  
End Sub
```



[Show All](#)

# AddNode Method

 [AddNode method as it applies to the \*\*DiagramNodeChildren\*\* object.](#)

Creates a diagram node. Returns a **DiagramNode** object that represents the new node.

*expression*.**AddNode**(*Index*, *nodeType*)

*expression* Required. An expression that returns a **DiagramNodeChildren** object

**Index** Optional **Variant**. The position of the node.

**nodeType** Optional [MsoDiagramNodeType](#). The type of node.

MsoDiagramNodeType can be one of these MsoDiagramNodeType constants.

**msoDiagramAssistant**

**msoDiagramNode** *default*

 [AddNode method as it applies to the \*\*DiagramNode\*\* object.](#)

Creates a diagram node. Returns a **DiagramNode** object that represents the new node. **DiagramNode** object.

*expression*.**AddNode**(*pos*, *nodeType*)

*expression* Required. An expression that returns a **DiagramNode** object.

**pos** Optional [MsoRelativeNodePosition](#). Where the node will be added, relative to another node.

MsoRelativeNodePosition can be one of these MsoRelativeNodePosition constants.

**msoAfterLastSibling**

**msoAfterNode** *default*

**msoBeforeFirstSibling**

**msoBeforeNode**

*nodeType* Optional [MsoDiagramNodeType](#). The type of node.

MsoDiagramNodeType can be one of these MsoDiagramNodeType constants.

**msoDiagramAssistant**

**msoDiagramNode** *default*

## Example

This example adds a node to a diagram node on the active sheet.

```
Sub DiagramNodeOBJ()  
  
    Dim nodDiagNode As DiagramNode  
    Dim shDiagram As Shape  
  
    Set shDiagram = ActiveSheet.Shapes.AddDiagram _  
        (Type:=msoDiagramOrgChart, _  
        Left:=10, _  
        Top:=15, _  
        Width:=400, _  
        Height:=475)  
  
    Set nodDiagNode = shDiagram.DiagramNode  
  
    'Add a root node to the diagram.  
    nodDiagNode.Children.AddNode  
  
End Sub
```



[Show All](#)

# AddNodes Method

*expression*.AddNodes(*SegmentType*, *EditingType*, *X1*, *Y1*, *X2*, *Y2*, *X3*, *Y3*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*SegmentType* Required [MsoSegmentType](#). The type of segment to be added.

MsoSegmentType can be one of these MsoSegmentType constants.

**msoSegmentLine**

**msoSegmentCurve**

*EditingType* Required [MsoEditingType](#). The editing property of the vertex.

MsoEditingType can be one of these MsoEditingType constants.

**msoEditingAuto**

**msoEditingCorner**

Cannot be **msoEditingSmooth** or **msoEditingSymmetric**

If *SegmentType* is **msoSegmentLine**, *EditingType* must be **msoEditingAuto**.

*X1* Required [Single](#).

If the *EditingType* of the new segment is **msoEditingAuto**, this argument specifies the horizontal distance (in points) from the upper-left corner of the document to the end point of the new segment.

If the *EditingType* of the new node is **msoEditingCorner**, this argument specifies the horizontal distance (in points) from the upper-left corner of the document to the first control point for the new segment.

*Y1* Required [Single](#).

If the *EditingType* of the new segment is **msoEditingAuto**, this argument specifies the horizontal distance (in points) from the upper-left corner of the document to the end point of the new segment.

If the *EditingType* of the new node is **msoEditingCorner**, this argument specifies the horizontal distance (in points) from the upper-left corner of the document to the first control point for the new segment.

## X2 Optional [Variant](#).

If the *EditingType* of the new segment is **msoEditingCorner**, this argument specifies the horizontal distance (in points) from the upper-left corner of the document to the second control point for the new segment.

If the *EditingType* of the new segment is **msoEditingAuto**, don't specify a value for this argument.

## Y2 Optional [Variant](#).

If the *EditingType* of the new segment is **msoEditingCorner**, this argument specifies the horizontal distance (in points) from the upper-left corner of the document to the second control point for the new segment.

If the *EditingType* of the new segment is **msoEditingAuto**, don't specify a value for this argument.

## X3 Optional [Variant](#).

If the *EditingType* of the new segment is **msoEditingCorner**, this argument specifies the horizontal distance (in points) from the upper-left corner of the document to the end point of the new segment.

If the *EditingType* of the new segment is **msoEditingAuto**, don't specify a

value for this argument.

**Y3** Optional [Variant](#).

If the ***EditingType*** of the new segment is **msoEditingCorner**, this argument specifies the vertical distance (in points) from the upper-left corner of the document to the end point of the new segment.

If the ***EditingType*** of the new segment is **msoEditingAuto**, don't specify a value for this argument.

## Example

This example adds a freeform with four segments to myDocument.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.BuildFreeform(msoEditingCorner, 360, 200)
    .AddNodes msoSegmentCurve, msoEditingCorner, _
        380, 230, 400, 250, 450, 300
    .AddNodes msoSegmentCurve, msoEditingAuto, 480, 200
    .AddNodes msoSegmentLine, msoEditingAuto, 480, 400
    .AddNodes msoSegmentLine, msoEditingAuto, 360, 200
    .ConvertToShape
End With
```



[Show All](#)

# AddPageItem Method

Adds an additional item to a multiple item page field.

*expression*.AddPageItem(*Item*, *ClearList*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*Item* Required **String**. Source name of a **PivotItem** object, corresponding to the specific [Online Analytical Processing \(OLAP\)](#) member unique name.

*ClearList* Optional **Variant**. If **False** (default), adds a page item to the existing list. If **True**, deletes all current items and adds *Item*.

## Remarks

To avoid run-time errors, the data source must be an OLAP source, the field chosen must currently be in the page position, and the [EnableMultiplePageItems](#) property must be set to **True**.

## Example

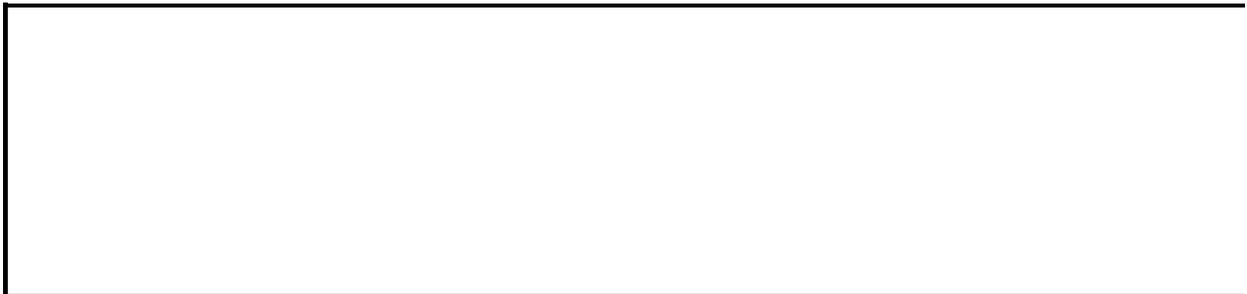
In this example, Microsoft Excel adds a page item with a source name titled "[Product].[All Products].[Food].[Eggs]". This example assumes an OLAP PivotTable exists on the active worksheet.

```
Sub UseAddPageItem()
```

```
    ' The source is an OLAP database and you can manually reorder it  
    ActiveSheet.PivotTables(1).CubeFields("[Product]"). _  
        EnableMultiplePageItems = True
```

```
    ' Add the page item titled "[Product].[All Products].[Food].[Egg  
    ActiveSheet.PivotTables(1).PivotFields("[Product]").AddPageItem  
        "[Product].[All Products].[Food].[Eggs]")
```

```
End Sub
```



[Show All](#)

# AddPicture Method

Creates a picture from an existing file. Returns a **Shape** object that represents the new picture.

*expression*.**AddPicture**(*FileName*, *LinkToFile*, *SaveWithDocument*, *Left*, *Top*, *Width*, *Height*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**FileName** Required **String**. The file from which the OLE object is to be created.

**LinkToFile** Required [MsoTriState](#). The file to link to.

MsoTriState can be one of these MsoTriState constants.

**msoCTrue**

**msoFalse** To make the picture an independent copy of the file.

**msoTriStateMixed**

**msoTriStateToggle**

**msoTrue** To link the picture to the file from which it was created.

**SaveWithDocument** Required [MsoTriState](#). To save the picture with the document.

MsoTriState can be one of these MsoTriState constants.

**msoCTrue**

**msoFalse** To store only the link information in the document.

**msoTriStateMixed**

**msoTriStateToggle**

**msoTrue** To save the linked picture with the document into which it's inserted. This argument must be **msoTrue** if **LinkToFile** is **msoFalse**.

**Left** Required **Single**. The position (in points) of the upper-left corner of the

picture relative to the upper-left corner of the document.

**Top** Required **Single**. The position (in points) of the upper-left corner of the picture relative to the top of the document.

**Width** Required **Single**. The width of the picture, in points.

**Height** Required **Single**. The height of the picture, in points.

## Example

This example adds a picture created from the file Music.bmp to myDocument. The inserted picture is linked to the file from which it was created and is saved with myDocument.

```
Set myDocument = Worksheets(1)
myDocument.Shapes.AddPicture _
    "c:\microsoft office\clipart\music.bmp", _
    True, True, 100, 100, 70, 70
```



[Show All](#)

# AddPolyline Method

Creates an open polyline or a closed polygon drawing. Returns a [Shape](#) object that represents the new polyline or polygon.

*expression*.**AddPolyline**(*SafeArrayOfPoints*)

*expression* Required. An expression that returns a **Shapes** object.

**SafeArrayOfPoints** Required **Variant**. An array of [coordinate pairs](#) that specifies the polyline drawing's vertices.

## Remarks

To form a closed polygon, assign the same coordinates to the first and last vertices in the polyline drawing.

## Example

This example adds a triangle to myDocument. Because the first and last points have the same coordinates, the polygon is closed and filled. The color of the triangle's interior will be the same as the default shape's fill color.

```
Dim triArray(1 To 4, 1 To 2) As Single
triArray(1, 1) = 25
triArray(1, 2) = 100
triArray(2, 1) = 100
triArray(2, 2) = 150
triArray(3, 1) = 150
triArray(3, 2) = 50
triArray(4, 1) = 25      ' Last point has same coordinates as first
triArray(4, 2) = 100
Set myDocument = Worksheets(1)
myDocument.Shapes.AddPolyline triArray
```



# AddReplacement Method

Adds an entry to the array of AutoCorrect replacements.

*expression*.**AddReplacement**(*What*, *Replacement*)

*expression* Required. An expression that returns an **AutoCorrect** object.

**What** Required **String**. The text to be replaced. If this string already exists in the array of AutoCorrect replacements, the existing substitute text is replaced by the new text.

**Replacement** Required **String**. The replacement text.

## Example

This example substitutes the word "Temp." for the word "Temperature" in the array of AutoCorrect replacements.

```
With Application.AutoCorrect  
    .AddReplacement "Temperature", "Temp."  
End With
```



[Show All](#)

# AddSet Method

Adds a new **CubeField** object to the **CubeFields** collection. The **CubeField** object corresponds to a set defined on the [Online Analytical Processing \(OLAP\)](#) provider for the cube.

*expression.AddSet(Name, Caption)*

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Name** Required **String**. A valid name in the SETS schema rowset.

**Caption** Required **String**. A string representing the field that will be displayed in the PivotTable view.

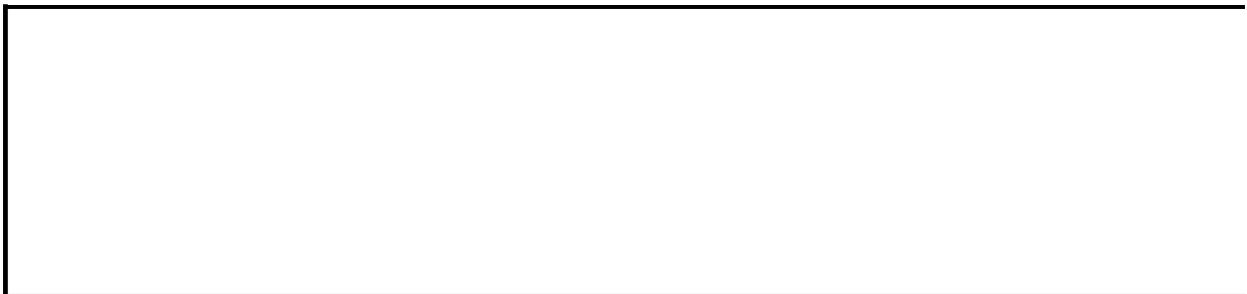
## Remarks

If a set with the name given in the argument *Name* does not exist, the **AddSet** method will return a run-time error.

## Example

In this example, Microsoft Excel adds a set titled "My Set" to the **CubeField** object. This example assumes an OLAP PivotTable report exists on the active worksheet. Also, this example assumes a field titled "Product" exists.

```
Sub UseAddSet()  
  
    Dim pvtOne As PivotTable  
    Dim strAdd As String  
    Dim strFormula As String  
    Dim cbfOne As CubeField  
  
    Set pvtOne = Sheet1.PivotTables(1)  
  
    strAdd = "[MySet]"  
    strFormula = "'{[Product].[All Products].[Food].children}'"  
  
    ' Establish connection with data source if necessary.  
    If Not pvtOne.PivotCache.IsConnected Then pvtOne.PivotCache.Make  
  
    ' Add a calculated member titled "[MySet]"  
    pvtOne.CalculatedMembers.Add Name:=strAdd, _  
        Formula:=strFormula, Type:=xlCalculatedSet  
  
    ' Add a set to the CubeField object.  
    Set cbfOne = pvtOne.CubeFields.AddSet(Name:="[MySet]", _  
        Caption:="My Set")  
  
End Sub
```



[Show All](#)

# AddShape Method

As it applies to the **Shapes** object, returns a [Shape](#) object that represents the new AutoShape in a worksheet. As it applies to the **CanvasShapes** object, returns a **Shape** object that represents the new AutoShape in a drawing canvas.

*expression.AddShape(Type, Left, Top, Width, Height)*

*expression* Required. An expression that returns a **Shapes** object.

**Type** Required [MsoAutoShapeType](#). Specifies the type of AutoShape to create.

MsoAutoShapeType can be one of these MsoAutoShapeType constants.

**msoShape16pointStar**

**msoShape24pointStar**

**msoShape32pointStar**

**msoShape4pointStar**

**msoShape5pointStar**

**msoShape8pointStar**

**msoShapeActionButtonBackorPrevious**

**msoShapeActionButtonBeginning**

**msoShapeActionButtonCustom**

**msoShapeActionButtonDocument**

**msoShapeActionButtonEnd**

**msoShapeActionButtonForwardorNext**

**msoShapeActionButtonHelp**

**msoShapeActionButtonHome**

**msoShapeActionButtonInformation**

**msoShapeActionButtonMovie**

**msoShapeActionButtonReturn**

**msoShapeActionButtonSound**

**msoShapeArc**

**msoShapeBalloon**  
**msoShapeBentArrow**  
**msoShapeBentUpArrow**  
**msoShapeBevel**  
**msoShapeBlockArc**  
**msoShapeCan**  
**msoShapeChevron**  
**msoShapeCircularArrow**  
**msoShapeCloudCallout**  
**msoShapeCross**  
**msoShapeCube**  
**msoShapeCurvedDownArrow**  
**msoShapeCurvedDownRibbon**  
**msoShapeCurvedLeftArrow**  
**msoShapeCurvedRightArrow**  
**msoShapeCurvedUpArrow**  
**msoShapeCurvedUpRibbon**  
**msoShapeDiamond**  
**msoShapeDonut**  
**msoShapeDoubleBrace**  
**msoShapeDoubleBracket**  
**msoShapeDoubleWave**  
**msoShapeDownArrow**  
**msoShapeDownArrowCallout**  
**msoShapeDownRibbon**  
**msoShapeExplosion1**  
**msoShapeExplosion2**  
**msoShapeFlowchartAlternateProcess**  
**msoShapeFlowchartCard**  
**msoShapeFlowchartCollate**  
**msoShapeFlowchartConnector**  
**msoShapeFlowchartData**  
**msoShapeFlowchartDecision**

**msoShapeFlowchartDelay**  
**msoShapeFlowchartDirectAccessStorage**  
**msoShapeFlowchartDisplay**  
**msoShapeFlowchartDocument**  
**msoShapeFlowchartExtract**  
**msoShapeFlowchartInternalStorage**  
**msoShapeFlowchartMagneticDisk**  
**msoShapeFlowchartManualInput**  
**msoShapeFlowchartManualOperation**  
**msoShapeFlowchartMerge**  
**msoShapeFlowchartMultidocument**  
**msoShapeFlowchartOffpageConnector**  
**msoShapeFlowchartOr**  
**msoShapeFlowchartPredefinedProcess**  
**msoShapeFlowchartPreparation**  
**msoShapeFlowchartProcess**  
**msoShapeFlowchartPunchedTape**  
**msoShapeFlowchartSequentialAccessStorage**  
**msoShapeFlowchartSort**  
**msoShapeFlowchartStoredData**  
**msoShapeFlowchartSummingJunction**  
**msoShapeFlowchartTerminator**  
**msoShapeFoldedCorner**  
**msoShapeHeart**  
**msoShapeHexagon**  
**msoShapeHorizontalScroll**  
**msoShapeIsoscelesTriangle**  
**msoShapeLeftArrow**  
**msoShapeLeftArrowCallout**  
**msoShapeLeftBrace**  
**msoShapeLeftBracket**  
**msoShapeLeftRightArrow**  
**msoShapeLeftRightArrowCallout**

**msoShapeLeftRightUpArrow**  
**msoShapeLeftUpArrow**  
**msoShapeLightningBolt**  
**msoShapeLineCallout1**  
**msoShapeLineCallout1AccentBar**  
**msoShapeLineCallout1BorderandAccentBar**  
**msoShapeLineCallout1NoBorder**  
**msoShapeLineCallout2**  
**msoShapeLineCallout2AccentBar**  
**msoShapeLineCallout2BorderandAccentBar**  
**msoShapeLineCallout2NoBorder**  
**msoShapeLineCallout3**  
**msoShapeLineCallout3AccentBar**  
**msoShapeLineCallout3BorderandAccentBar**  
**msoShapeLineCallout3NoBorder**  
**msoShapeLineCallout4**  
**msoShapeLineCallout4AccentBar**  
**msoShapeLineCallout4BorderandAccentBar**  
**msoShapeLineCallout4NoBorder**  
**msoShapeMixed**  
**msoShapeMoon**  
**msoShapeNoSymbol**  
**msoShapeNotchedRightArrow**  
**msoShapeNotPrimitive**  
**msoShapeOctagon**  
**msoShapeOval**  
**msoShapeOvalCallout**  
**msoShapeParallelogram**  
**msoShapePentagon**  
**msoShapePlaque**  
**msoShapeQuadArrow**  
**msoShapeQuadArrowCallout**  
**msoShapeRectangle**

**msoShapeRectangularCallout**  
**msoShapeRegularPentagon**  
**msoShapeRightArrow**  
**msoShapeRightArrowCallout**  
**msoShapeRightBrace**  
**msoShapeRightBracket**  
**msoShapeRightTriangle**  
**msoShapeRoundedRectangle**  
**msoShapeRoundedRectangularCallout**  
**msoShapeSmileyFace**  
**msoShapeStripedRightArrow**  
**msoShapeSun**  
**msoShapeTrapezoid**  
**msoShapeUpArrow**  
**msoShapeUpArrowCallout**  
**msoShapeUpDownArrow**  
**msoShapeUpDownArrowCallout**  
**msoShapeUpRibbon**  
**msoShapeUTurnArrow**  
**msoShapeVerticalScroll**  
**msoShapeWave**

***Left*** , ***Top*** Required **Single**. The position (in points) of the upper-left corner of the AutoShape's bounding box relative to the upper-left corner of the document.

***Width*** , ***Height*** Required **Single**. The width and height of the AutoShape's bounding box, in points.

## Remarks

To change the type of an *AutoShape* that you've added, set the [AutoShapeType](#) property.

## Example

This example adds a rectangle to myDocument.

```
Set myDocument = Worksheets(1)  
myDocument.Shapes.AddShape msoShapeRectangle, 50, 50, 100, 200
```



[Show All](#)

# AddTextbox Method

Creates a text box. Returns a [Shape](#) object that represents the new text box.

*expression*.AddTextbox(***Orientation***, ***Left***, ***Top***, ***Width***, ***Height***)

*expression* Required. An expression that returns one of the objects in the Applies To list.

***Orientation*** Required [MsoTextOrientation](#). The orientation of the textbox.

MsoTextOrientation can be one of these MsoTextOrientation constants.

**msoTextOrientationDownward**

**msoTextOrientationHorizontal**

**msoTextOrientationHorizontalRotatedFarEast**

**msoTextOrientationMixed**

**msoTextOrientationUpward**

**msoTextOrientationVertical**

**msoTextOrientationVerticalFarEast**

Some of these constants may not be available to you, depending on the language support (U.S. English, for example) that you've selected or installed.

***Left*** Required **Single**. The position (in points) of the upper-left corner of the text box relative to the upper-left corner of the document.

***Top*** Required **Single**. The position (in points) of the upper-left corner of the text box relative to the top of the document.

***Width*** Required **Single**. The width of the text box, in points.

***Height*** Required **Single**. The height of the text box, in points.

## Example

This example adds a text box that contains the text "Test Box" to myDocument.

```
Set myDocument = Worksheets(1)
myDocument.Shapes.AddTextbox(msoTextOrientationHorizontal, _
    100, 100, 200, 50) _
    .TextFrame.Characters.Text = "Test Box"
```



[Show All](#)

# AddTextEffect Method

Creates a WordArt object. Returns a [Shape](#) object that represents the new WordArt object.

*expression.AddTextEffect(PresetTextEffect, Text, FontName, FontSize, FontBold, FontItalic, Left, Top)*

*expression* Required. An expression that returns one of the objects in the Applies To list.

*PresetTextEffect* Required [MsoPresetTextEffect](#). The preset text effect.

MsoPresetTextEffect can be one of these MsoPresetTextEffect constants.

**msoTextEffect1**

**msoTextEffect2**

**msoTextEffect3**

**msoTextEffect4**

**msoTextEffect5**

**msoTextEffect6**

**msoTextEffect7**

**msoTextEffect8**

**msoTextEffect9**

**msoTextEffect10**

**msoTextEffect11**

**msoTextEffect12**

**msoTextEffect13**

**msoTextEffect14**

**msoTextEffect15**

**msoTextEffect16**

**msoTextEffect17**

**msoTextEffect18**

**msoTextEffect19**

**msoTextEffect20**  
**msoTextEffect21**  
**msoTextEffect22**  
**msoTextEffect23**  
**msoTextEffect24**  
**msoTextEffect25**  
**msoTextEffect26**  
**msoTextEffect27**  
**msoTextEffect28**  
**msoTextEffect29**  
**msoTextEffect30**  
**msoTextEffectMixed**

**Text** Required **String**. The text in the WordArt.

**FontName** Required **String**. The name of the font used in the WordArt.

**FontSize** Required **Single**. The size (in points) of the font used in the WordArt.

**FontBold** Required **MsoTriState**. The font used in the WordArt to bold.

MsoTriState can be one of these MsoTriState constants.

**msoCTrue**  
**msoFalse**  
**msoTriStateMixed**  
**msoTriStateToggle**  
**msoTrue**

**FontItalic** Required **MsoTriState**. The font used in the WordArt to italic.

MsoTriState can be one of these MsoTriState constants.

**msoCTrue**  
**msoFalse**  
**msoTriStateMixed**  
**msoTriStateToggle**

## **msoTrue**

**Left** Required **Single**. The position (in points) of the upper-left corner of the WordArt's bounding box relative to the upper-left corner of the document.

**Top** Required **Single**. The position (in points) of the upper-left corner of the WordArt's bounding box relative to the top of the document.

## **Remarks**

When you add WordArt to a document, the height and width of the WordArt are automatically set based on the size and amount of text you specify.

## Example

This example adds WordArt that contains the text "Test" to myDocument.

```
Set myDocument = Worksheets(1)
Set newWordArt = myDocument.Shapes.AddTextEffect( _
    PresetTextEffect:=msoTextEffect1, Text:="Test", _
    FontName:="Arial Black", FontSize:=36, _
    FontBold:=msoFalse, FontItalic:=msoFalse, Left:=10, _
    Top:=10)
```



# AddToFavorites Method

Adds a shortcut to the workbook or hyperlink to the Favorites folder.

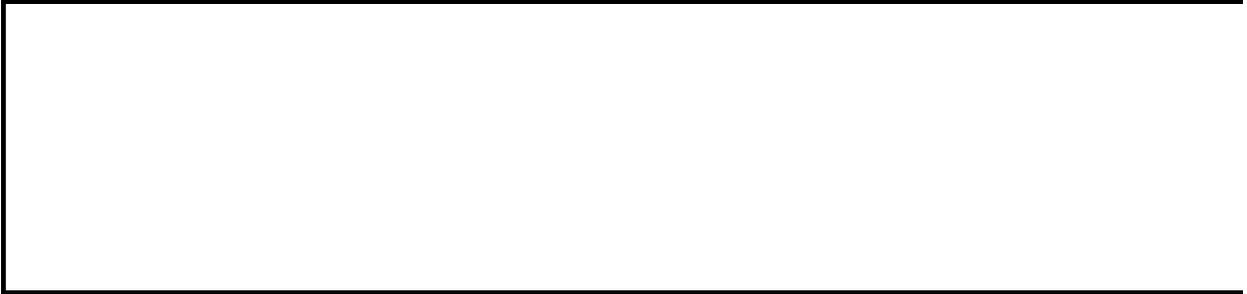
*expression*.**AddToFavorites**

*expression* Required. An expression that returns a **Workbook** or **Hyperlink** object.

## Example

This example adds a shortcut to the active workbook to the Favorites folder.

ActiveWorkbook.**AddToFavorites**



[Show All](#)

# AdvancedFilter Method

Filters or copies data from a list based on a criteria range. If the initial selection is a single cell, that cell's current region is used. **Variant**.

*expression*.**AdvancedFilter**(*Action*, *CriteriaRange*, *CopyToRange*, *Unique*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Action** Required [XIFilterAction](#).

XIFilterAction can be one of these XIFilterAction constants.

**xIFilterCopy**

**xIFilterInPlace**

**CriteriaRange** Optional **Variant**. The criteria range. If this argument is omitted, there are no criteria.

**CopyToRange** Optional **Variant**. The destination range for the copied rows if **Action** is **xIFilterCopy**. Otherwise, this argument is ignored.

**Unique** Optional **Variant**. **True** to filter unique records only. **False** to filter all records that meet the criteria. The default value is **False**.

## Example

This example filters a database (named "Database") based on a criteria range named "Criteria."

```
Range("Database").AdvancedFilter _  
    Action:=xlFilterInPlace, _  
    CriteriaRange:=Range("Criteria")
```



[Show All](#)

# Align Method

Aligns the shapes in the specified range of shapes.

*expression*.Align(**AlignCmd**, **RelativeTo**)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**AlignCmd** Required [MsoAlignCmd](#). Specifies the way the shapes in the specified shape range are to be aligned.

MsoAlignCmd can be one of these MsoAlignCmd constants.

**msoAlignCenters**

**msoAlignMiddles**

**msoAlignTops**

**msoAlignBottoms**

**msoAlignLefts**

**msoAlignRights**

**RelativeTo** Required [MsoTriState](#). Not used in Microsoft Excel. Must be **False**.

MsoTriState can be one of these MsoTriState constants.

**msoCTrue**

**msoFalse**

**msoTriStateMixed**

**msoTriStateToggle**

**msoTrue**

## Example

This example aligns the left edges of all the shapes in the specified range in myDocument with the left edge of the leftmost shape in the range.

```
Set myDocument = Worksheets(1)
myDocument.Shapes.SelectAll
Selection.ShapeRange.Align msoAlignLefts, False
```



# Apply Method

Applies to the specified shape formatting that's been copied by using the [PickUp](#) method.

*expression*.**Apply**

*expression* Required. An expression that returns a **Shape** or **ShapeRange** object.

## Example

This example copies the formatting of shape one on myDocument and then applies the copied formatting to shape two.

```
Set myDocument = Worksheets(1)
With myDocument
    .Shapes(1).PickUp
    .Shapes(2).Apply
End With
```



[Show All](#)

# ApplyCustomType Method

 [ApplyCustomType](#) method as it applies to the **Series** object.

Applies a standard or custom chart type to a series.

*expression*.**ApplyCustomType**(*ChartType*)

*expression* Required. An expression that returns one of the above objects.

*ChartType* Required [XLChartType](#). A standard chart type.

XlChartType can be one of these XlChartType constants.

**xlLine**

**xlLineMarkersStacked**

**xlLineStacked**

**xlPie**

**xlPieOfPie**

**xlPyramidBarStacked**

**xlPyramidCol**

**xlPyramidColClustered**

**xlPyramidColStacked**

**xlPyramidColStacked100**

**xlRadar**

**xlRadarFilled**

**xlRadarMarkers**

**xlStockHLC**

**xlStockOHLC**

**xlStockVHLC**

**xlStockVOHLC**

**xlSurface**

**xlSurfaceTopView**

**xlSurfaceTopViewWireframe**

**xlSurfaceWireframe**  
**xlXYScatter**  
**xlXYScatterLines**  
**xlXYScatterLinesNoMarkers**  
**xlXYScatterSmooth**  
**xlXYScatterSmoothNoMarkers**  
**xl3DArea**  
**xl3DAreaStacked**  
**xl3DAreaStacked100**  
**xl3DBarClustered**  
**xl3DBarStacked**  
**xl3DBarStacked100**  
**xl3DColumn**  
**xl3DColumnClustered**  
**xl3DColumnStacked**  
**xl3DColumnStacked100**  
**xl3DLine**  
**xl3DPie**  
**xl3DPieExploded**  
**xlArea**  
**xlAreaStacked**  
**xlAreaStacked100**  
**xlBarClustered**  
**xlBarOfPie**  
**xlBarStacked**  
**xlBarStacked100**  
**xlBubble**  
**xlBubble3DEffect**  
**xlColumnClustered**  
**xlColumnStacked**  
**xlColumnStacked100**  
**xlConeBarClustered**  
**xlConeBarStacked**

**xlConeBarStacked100**  
**xlConeCol**  
**xlConeColClustered**  
**xlConeColStacked**  
**xlConeColStacked100**  
**xlCylinderBarClustered**  
**xlCylinderBarStacked**  
**xlCylinderBarStacked100**  
**xlCylinderCol**  
**xlCylinderColClustered**  
**xlCylinderColStacked**  
**xlCylinderColStacked100**  
**xlDoughnut**  
**xlDoughnutExploded**  
**xlLineMarkers**  
**xlLineMarkersStacked100**  
**xlLineStacked100**  
**xlPieExploded**  
**xlPyramidBarClustered**  
**xlPyramidBarStacked100**

[ApplyCustomType](#) method as it applies to the **Chart** object.

Applies a standard or custom chart type to a chart.

*expression*.**ApplyCustomType**(*ChartType*, *TypeName*)

*expression* Required. An expression that returns one of the above objects.

**Chart Type** Required [xlChartType](#). A standard chart type.

xlChartType can be one of these xlChartType constants.

**xlLine**  
**xlLineMarkersStacked**  
**xlLineStacked**

**xlPie**  
**xlPieOfPie**  
**xlPyramidBarStacked**  
**xlPyramidCol**  
**xlPyramidColClustered**  
**xlPyramidColStacked**  
**xlPyramidColStacked100**  
**xlRadar**  
**xlRadarFilled**  
**xlRadarMarkers**  
**xlStockHLC**  
**xlStockOHLC**  
**xlStockVHLC**  
**xlStockVOHLC**  
**xlSurface**  
**xlSurfaceTopView**  
**xlSurfaceTopViewWireframe**  
**xlSurfaceWireframe**  
**xlXYScatter**  
**xlXYScatterLines**  
**xlXYScatterLinesNoMarkers**  
**xlXYScatterSmooth**  
**xlXYScatterSmoothNoMarkers**  
**xl3DArea**  
**xl3DAreaStacked**  
**xl3DAreaStacked100**  
**xl3DBarClustered**  
**xl3DBarStacked**  
**xl3DBarStacked100**  
**xl3DColumn**  
**xl3DColumnClustered**  
**xl3DColumnStacked**  
**xl3DColumnStacked100**

**xl3DLine**  
**xl3DPie**  
**xl3DPieExploded**  
**xlArea**  
**xlAreaStacked**  
**xlAreaStacked100**  
**xlBarClustered**  
**xlBarOfPie**  
**xlBarStacked**  
**xlBarStacked100**  
**xlBubble**  
**xlBubble3DEffect**  
**xlColumnClustered**  
**xlColumnStacked**  
**xlColumnStacked100**  
**xlConeBarClustered**  
**xlConeBarStacked**  
**xlConeBarStacked100**  
**xlConeCol**  
**xlConeColClustered**  
**xlConeColStacked**  
**xlConeColStacked100**  
**xlCylinderBarClustered**  
**xlCylinderBarStacked**  
**xlCylinderBarStacked100**  
**xlCylinderCol**  
**xlCylinderColClustered**  
**xlCylinderColStacked**  
**xlCylinderColStacked100**  
**xlDoughnut**  
**xlDoughnutExploded**  
**xlLineMarkers**  
**xlLineMarkersStacked100**

**xlLineStacked100**

**xlPieExploded**

**xlPyramidBarClustered**

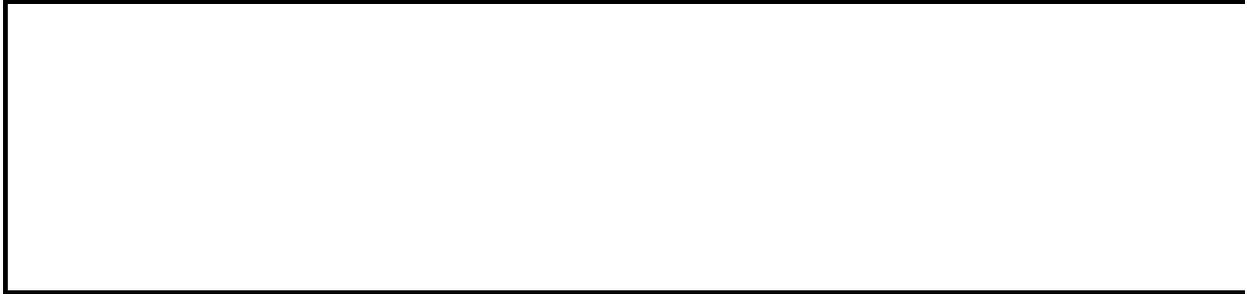
**xlPyramidBarStacked100**

***TypeName*** Optional **Variant** (used only with a **Chart** object). The name of the custom chart type if ***ChartType*** specifies a custom chart gallery.

## Example

This example applies the “Line with Data Markers” chart type to chart one.

```
Charts(1).ApplyCustomType xlLineMarkers
```



[Show All](#)

# ApplyDataLabels Method

Applies data labels to a point, a series, or all the series in a chart.

*expression*.**ApplyDataLabels**(*Type*, *LegendKey*, *AutoText*, *HasLeaderLines*, *ShowSeriesName*, *ShowCategoryName*, *ShowValue*, *ShowPercentage*, *ShowBubbleSize*, *Separator*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Type** Optional [XLDataLabelsType](#). The type of data label to apply.

XLDataLabelsType can be one of these XLDataLabelsType constants.

**xlDataLabelsShowBubbleSizes**

**xlDataLabelsShowLabelAndPercent**. Percentage of the total, and category for the point. Available only for pie charts and doughnut charts.

**xlDataLabelsShowPercent**. Percentage of the total. Available only for pie charts and doughnut charts.

**xlDataLabelsShowLabel**. Category for the point.

**xlDataLabelsShowNone**. No data labels.

**xlDataLabelsShowValue**. *default*. Value for the point (assumed if this argument isn't specified).

**LegendKey** Optional **Variant**. **True** to show the legend key next to the point. The default value is **False**.

**AutoText** Optional **Variant**. **True** if the object automatically generates appropriate text based on content.

**HasLeaderLines** Optional **Variant**. For the [Chart](#) and [Series](#) objects, **True** if the series has leader lines.

**ShowSeriesName** Optional **Variant**. The series name for the data label.

**ShowCategoryName** Optional **Variant**. The category name for the data label.

**ShowValue** Optional **Variant**. The value for the data label.

**ShowPercentage** Optional **Variant**. The percentage for the data label.

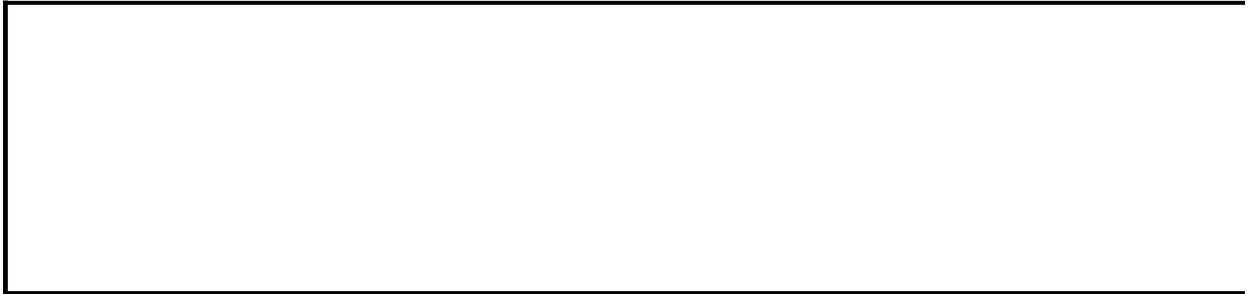
**ShowBubbleSize** Optional **Variant**. The bubble size for the data label.

**Separator** Optional **Variant**. The separator for the data label.

## Example

This example applies category labels to series one in Chart1.

```
Charts("Chart1").SeriesCollection(1). _  
    ApplyDataLabels Type:=xlDataLabelsShowLabel
```



[Show All](#)

# ApplyNames Method

Applies names to the cells in the specified range.

*expression*.**ApplyNames**(*Names*, *IgnoreRelativeAbsolute*,  
*UseRowColumnNames*, *OmitColumn*, *OmitRow*, *Order*, *AppendLast*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Names** Optional **VARIANT**. An array of the names to be applied. If this argument is omitted, all names on the sheet are applied to the range.

**IgnoreRelativeAbsolute** Optional **VARIANT**. **True** to replace references with names, regardless of the reference types of either the names or references. **False** to replace absolute references only with absolute names, relative references only with relative names, and mixed references only with mixed names. The default value is **True**.

**UseRowColumnNames** Optional **VARIANT**. **True** to use the names of row and column ranges that contain the specified range if names for the range cannot be found. **False** to ignore the **OmitColumn** and **OmitRow** arguments. The default value is **True**.

**OmitColumn** Optional **VARIANT**. **True** to replace the entire reference with the row-oriented name. The column-oriented name can be omitted only if the referenced cell is in the same column as the formula and is within a row-oriented named range. The default value is **True**.

**OmitRow** Optional **VARIANT**. **True** to replace the entire reference with the column-oriented name. The row-oriented name can be omitted only if the referenced cell is in the same row as the formula and is within a column-oriented named range. The default value is **True**.

**Order** Optional [XlApplyNamesOrder](#). Determines which range name is listed first when a cell reference is replaced by a row-oriented and column-oriented range name.

XlApplyNamesOrder can be one of these XlApplyNamesOrder constants.

**xlColumnThenRow**

**xlRowThenColumn** *default*

**AppendLast** Optional **Variant**. **True** to replace the definitions of the names in **Names** and also replace the definitions of the last names that were defined. **False** to replace the definitions of the names in **Names** only. The default value is **False**.

## Remarks

You can use the **Array** function to create the list of names for the *Names* argument.

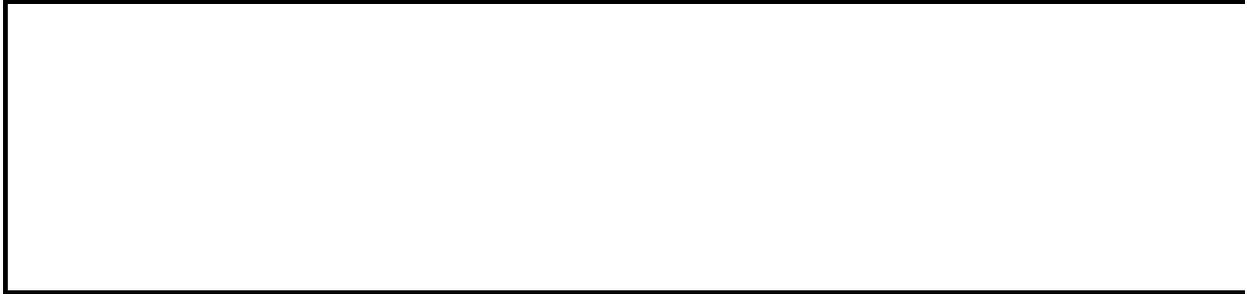
If you want to apply names to the entire sheet, use `Cells.ApplyNames`.

You cannot "unapply" names; to delete names, use the **Delete** method.

## Example

This example applies names to the entire sheet.

```
Cells.ApplyNames Names:=Array("Sales", "Profits")
```



# ApplyOutlineStyles Method

Applies outlining styles to the specified range.

*expression*.**ApplyOutlineStyles**

*expression* Required. An expression that returns a **Range** object.

## Example

The following example applies automatic outlining styles to the selection. The selection must include the entire outline range on a worksheet.

`Selection.ApplyOutlineStyles`



# AreaGroups Method

On a 2-D chart, returns an object that represents either a single area chart group (a [ChartGroup](#) object) or a collection of the area chart groups (a [ChartGroups](#) collection).

*expression*.AreaGroups(*Index*)

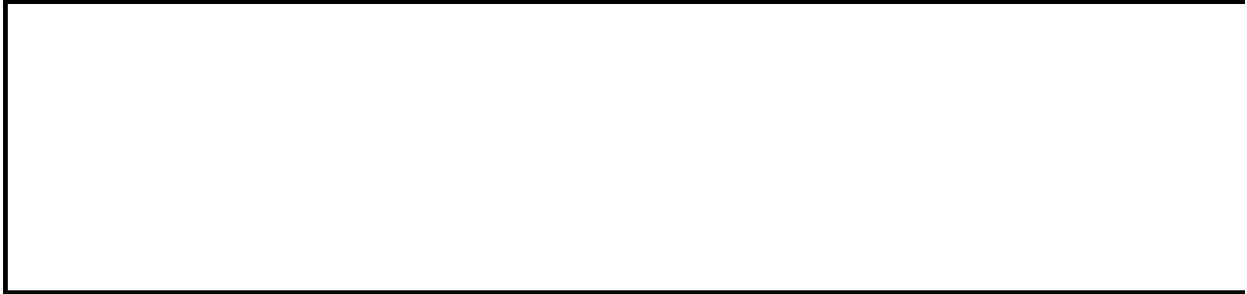
*expression* Required. An expression that returns a **Chart** object.

*Index* Optional **Variant**. The chart group number.

## Example

This example turns on drop lines for the 2-D area chart group.

```
Charts(1).AreaGroups(1).HasDropLines = True
```



[Show All](#)

# Arrange Method

Arranges the windows on the screen. **Variant**.

*expression*.**Arrange**(*ArrangeStyle*, *ActiveWorkbook*, *SyncHorizontal*, *SyncVertical*)

*expression* Required. An expression that returns one of the objects in the Applies To list

*ArrangeStyle* Optional [XlArrangeStyle](#).

XlArrangeStyle can be one of these XlArrangeStyle constants.

**xlArrangeStyleCascade**. Windows are cascaded.

**xlArrangeStyleTiled** *default*. Windows are tiled

**xlArrangeStyleHorizontal**. Windows are arranged horizontally.

**xlArrangeStyleVertical**. Windows are arranged vertically.

**ActiveWorkbook** Optional **Variant**. **True** to arrange only the visible windows of the active workbook. **False** to arrange all windows. The default value is **False**.

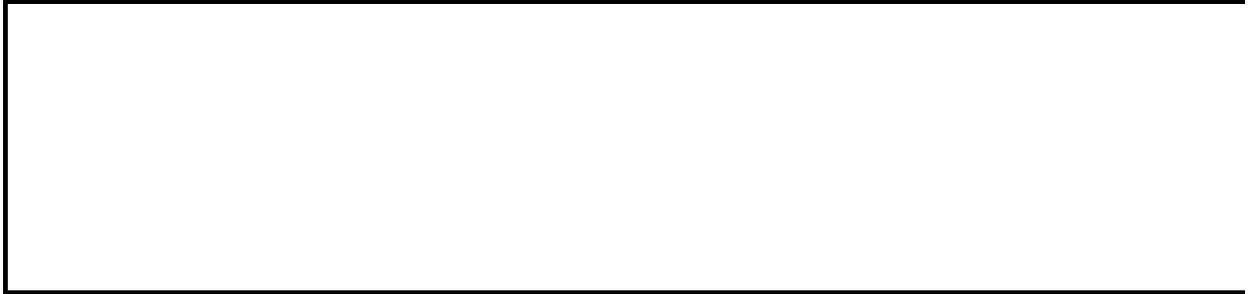
**SyncHorizontal** Optional **Variant**. Ignored if **ActiveWorkbook** is **False** or omitted. **True** to synchronize the windows of the active workbook when scrolling horizontally. **False** to not synchronize the windows. The default value is **False**.

**SyncVertical** Optional **Variant**. Ignored if **ActiveWorkbook** is **False** or omitted. **True** to synchronize the windows of the active workbook when scrolling vertically. **False** to not synchronize the windows. The default value is **False**.

## Example

This example tiles all the windows in the application.

```
Application.Windows.Arrange ArrangeStyle:=xlArrangeStyleTiled
```



# AutoComplete Method

Returns an AutoComplete match from the list. If there's no AutoComplete match or if more than one entry in the list matches the string to complete, this method returns an empty string.

*expression*.**AutoComplete**(*String*)

*expression* Required. An expression that returns a **Range** object (must be a single cell).

**String** Required **String**. The string to complete.

## Remarks

This method works even if the AutoComplete feature is disabled.

## Example

This example returns the AutoComplete match for the string segment "Ap." An AutoComplete match is made if the column containing cell A5 contains a contiguous list and one of the entries in the list contains a match for the string.

```
s = Worksheets(1).Range("A5").AutoComplete("Ap")
If Len(s) > 0 Then
    MsgBox "Completes to " & s
Else
    MsgBox "Has no completion"
End If
```



[Show All](#)

# AutoFill Method

Performs an autofill on the cells in the specified range. **Variant**.

*expression*.**AutoFill**(*Destination*, *Type*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Destination** Required **Range** object. The cells to be filled. The destination must include the source range.

**Type** Optional [XlAutoFillType](#). Specifies the fill type.

XlAutoFillType can be one of these XlAutoFillType constants.

**xlFillDays**

**xlFillFormats**

**xlFillSeries**

**xlFillWeekdays**

**xlGrowthTrend**

**xlFillCopy**

**xlFillDefault** *default*

**xlFillMonths**

**xlFillValues**

**xlFillYears**

**xlLinearTrend**

If this argument is **xlFillDefault** or omitted, Microsoft Excel selects the most appropriate fill type, based on the source range.

## Example

This example performs an autofill on cells A1:A20 on Sheet1, based on the source range A1:A2 on Sheet1. Before running this example, type **1** in cell A1 and type **2** in cell A2.

```
Set sourceRange = Worksheets("Sheet1").Range("A1:A2")  
Set fillRange = Worksheets("Sheet1").Range("A1:A20")  
sourceRange.AutoFill Destination:=fillRange
```



[Show All](#)

# AutoFilter Method

Filters a list using the AutoFilter. **Variant**.

**Note** Apply the [AutoFilter](#) property to a **Worksheet** object to return an [AutoFilter](#) object.

*expression*.AutoFilter(**Field**, **Criteria1**, **Operator**, **Criteria2**, **VisibleDropDown**)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Field** Optional **Variant**. The integer offset of the field on which you want to base the filter (from the left of the list; the leftmost field is field one).

**Criteria1** Optional **Variant**. The criteria (a string; for example, "101"). Use "=" to find blank fields, or use "<>" to find nonblank fields. If this argument is omitted, the criteria is All. If **Operator** is **xlTop10Items**, **Criteria1** specifies the number of items (for example, "10").

**Operator** Optional [XlAutoFilterOperator](#).

XlAutoFilterOperator can be one of these XlAutoFilterOperator constants.

**xlAnd** *default*

**xlBottom10Items**

**xlBottom10Percent**

**xlOr**

**xlTop10Items**

**xlTop10Percent**

Use **xlAnd** and **xlOr** with **Criteria1** and **Criteria2** to construct compound criteria.

**Criteria2** Optional **Variant**. The second criteria (a string). Used with **Criteria1** and **Operator** to construct compound criteria.

**VisibleDropDown** Optional **Variant**. **True** to display the AutoFilter drop-

down arrow for the filtered field. **False** to hide the AutoFilter drop-down arrow for the filtered field. **True** by default.

## Remarks

If you omit all the arguments, this method simply toggles the display of the AutoFilter drop-down arrows in the specified range.

## Example

This example filters a list starting in cell A1 on Sheet1 to display only the entries in which field one is equal to the string "Otis". The drop-down arrow for field one will be hidden.

```
Worksheets("Sheet1").Range("A1").AutoFilter _  
    field:=1, _  
    Criteria1:="Otis"  
    VisibleDropDown:=False
```



# AutoFit Method

Changes the width of the columns in the range or the height of the rows in the range to achieve the best fit.

*expression*.**AutoFit**

*expression* Required. An expression that returns a **Range** object. Must be a row or a range of rows, or a column or a range of columns. Otherwise, this method generates an error.

## **Remarks**

One unit of column width is equal to the width of one character in the Normal style.

## Example

This example changes the width of columns A through I on Sheet1 to achieve the best fit.

```
Worksheets("Sheet1").Columns("A:I").AutoFit
```

This example changes the width of columns A through E on Sheet1 to achieve the best fit, based only on the contents of cells A1:E1.

```
Worksheets("Sheet1").Range("A1:E1").Columns.AutoFit
```



[Show All](#)

# AutoFormat Method

 [AutoFormat method as it applies to the \*\*Range\*\* object.](#)

Automatically formats the specified range, using a predefined format.

*expression*.**AutoFormat**(*Format*, *Number*, *Font*, *Alignment*, *Border*, *Pattern*, *Width*)

*expression* Required. An expression that returns one of the above objects.

**Format** Optional [XlRangeAutoFormat](#). The specified AutoFormat.

XlRangeAutoFormat can be one of these XlRangeAutoFormat constants.

**xlRangeAutoFormat3DEffects1**

**xlRangeAutoFormat3DEffects2**

**xlRangeAutoFormatAccounting1**

**xlRangeAutoFormatAccounting2**

**xlRangeAutoFormatAccounting3**

**xlRangeAutoFormatAccounting4**

**xlRangeAutoFormatClassic1** *default*

**xlRangeAutoFormatClassic2**

**xlRangeAutoFormatClassic3**

**xlRangeAutoFormatClassicPivotTable**

**xlRangeAutoFormatColor1**

**xlRangeAutoFormatColor2**

**xlRangeAutoFormatColor3**

**xlRangeAutoFormatList1**

**xlRangeAutoFormatList2**

**xlRangeAutoFormatList3**

**xlRangeAutoFormatLocalFormat1**

**xlRangeAutoFormatLocalFormat2**

**xlRangeAutoFormatLocalFormat3**

**xlRangeAutoFormatLocalFormat4**  
**xlRangeAutoFormatNone**  
**xlRangeAutoFormatPTNone**  
**xlRangeAutoFormatReport1**  
**xlRangeAutoFormatReport10**  
**xlRangeAutoFormatReport2**  
**xlRangeAutoFormatReport3**  
**xlRangeAutoFormatReport4**  
**xlRangeAutoFormatReport5**  
**xlRangeAutoFormatReport6**  
**xlRangeAutoFormatReport7**  
**xlRangeAutoFormatReport8**  
**xlRangeAutoFormatReport9**  
**xlRangeAutoFormatSimple**  
**xlRangeAutoFormatTable1**  
**xlRangeAutoFormatTable10**  
**xlRangeAutoFormatTable2**  
**xlRangeAutoFormatTable3**  
**xlRangeAutoFormatTable4**  
**xlRangeAutoFormatTable5**  
**xlRangeAutoFormatTable6**  
**xlRangeAutoFormatTable7**  
**xlRangeAutoFormatTable8**  
**xlRangeAutoFormatTable9**

The default constant is **xlRangeAutoFormatClassic1**. Some of these constants may not be available to you, depending on the language support (U.S. English, for example) that you've selected or installed.

**Number** Optional **Variant**. **True** to include number formats in the AutoFormat. The default value is **True**.

**Font** Optional **Variant**. **True** to include font formats in the AutoFormat. The default value is **True**.

**Alignment** Optional **Variant**. **True** to include alignment in the AutoFormat. The default value is **True**.

**Border** Optional **Variant**. **True** to include border formats in the AutoFormat. The default value is **True**.

**Pattern** Optional **Variant**. **True** to include pattern formats in the AutoFormat. The default value is **True**.

**Width** Optional **Variant**. **True** to include column width and row height in the AutoFormat. The default value is **True**.



[AutoFormat method as it applies to the \*\*Chart\*\* object.](#)

Automatically formats the specified chart.

*expression*.**AutoFormat**(*Gallery*, *Format*)

*expression* Required. An expression that returns one of the above objects.

**Gallery** Required **Long**. The specified Gallery.

**Format** Optional **Variant**. The specified AutoFormat.

## Remarks

If the range is a single cell, this method also formats the active region surrounding the cell. In other words, the following two statements are equivalent:

```
Cells("A1").AutoFormat  
Cells("A1").CurrentRegion.AutoFormat
```

## Example

This example formats cells A1:D8 on Sheet1, using a predefined format.

```
Worksheets("Sheet1").Range("A1:D8").  
    AutoFormat Format:=xlRangeAutoFormatClassic1
```



# AutomaticLength Method

Specifies that the first segment of the callout line (the segment attached to the text callout box) be scaled automatically when the callout is moved. Use the [CustomLength](#) method to specify that the first segment of the callout line retain the fixed length returned by the [Length](#) property whenever the callout is moved. Applies only to callouts whose lines consist of more than one segment (types **msoCalloutThree** and **msoCalloutFour**).

*expression*.**AutomaticLength**

*expression* Required. An expression that returns a **CalloutFormat** object.

## Remarks

Applying this method sets the [AutoLength](#) property to **True**.

## Example

This example toggles between an automatically scaling first segment and one with a fixed length for the callout line for shape one on myDocument. For the example to work, shape one must be a callout.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(1).Callout
    If .AutoLength Then
        .CustomLength 50
    Else
        .AutomaticLength
    End If
End With
```



# AutoOutline Method

Automatically creates an outline for the specified range. If the range is a single cell, Microsoft Excel creates an outline for the entire sheet. The new outline replaces any existing outline.

*expression*.**AutoOutline**

*expression* Required. An expression that returns a **Range** object.

## Example

This example creates an outline for the range A1:G37 on Sheet1. The range must contain either a summary row or a summary column.

```
Worksheets("Sheet1").Range("A1:G37").AutoOutline
```



# AutoShow Method

Displays the number of top or bottom items for a row, page, or column field in the specified PivotTable report.

*expression*.**AutoShow**(*Type*, *Range*, *Count*, *Field*)

*expression* Required. An expression that returns one of the objects in the Applies To list

**Type** Required **Long**. Use **xlAutomatic** to cause the specified PivotTable report to show the items that match the specified criteria. Use **xlManual** to disable this feature.

**Range** Required **Long**. The location at which to start showing items. Can be either of the following constants: **xlTop** or **xlBottom**.

**Count** Required **Long**. The number of items to be shown.

**Field** Required **String**. The name of the base data field. You must specify the unique name (as returned from the [SourceName](#) property), and not the displayed name.

## Example

This example shows only the top two companies, based on the sum of sales:

```
ActiveSheet.PivotTables("Pivot1").PivotFields("Company") _  
    .AutoShow xlAutomatic, xlTop, 2, "Sum of Sales"
```



[Show All](#)

# AutoSort Method

Establishes automatic field-sorting rules for PivotTable reports.

*expression*.**AutoSort**(*Order*, *Field*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Order** Required [XISortOrder](#). The sort order.

XISortOrder can be one of these XISortOrder constants.

**xlAscending**

**xlDescending**

**xlManual**. To disable automatic sorting.

**Field** Required **String**. The name of the sort key field. You must specify the unique name (as returned from the **SourceName** property), and not the displayed name.

## Example

This example sorts the Company field in descending order, based on the sum of sales.

```
ActiveSheet.PivotTables(1).PivotField("Company") _  
    .AutoSort xlDescending, "Sum of Sales"
```



[Show All](#)

# Axes Method

Returns an object that represents either a single axis or a collection of the axes on the chart.

*expression*.**Axes**(*Type*, *AxisGroup*)

*expression* Required. An expression that returns a **Chart** object.

**Type** Optional **Variant**. Specifies the axis to return. Can be one of the following **XlAxisType** constants: **xlValue**, **xlCategory**, or **xlSeriesAxis** (**xlSeriesAxis** is valid only for 3-D charts).

**AxisGroup** Optional **XlAxisGroup**. Specifies the axis group. If this argument is omitted, the primary group is used. 3-D charts have only one axis group.

XlAxisGroup can be one of these XlAxisGroup constants.

**xlPrimary** *default*

**xlSecondary**

## Example

This example adds an axis label to the category axis in Chart1.

```
With Charts("Chart1").Axes(xlCategory)
    .HasTitle = True
    .AxisTitle.Text = "July Sales"
End With
```

This example turns off major gridlines for the category axis in Chart1.

```
Charts("Chart1").Axes(xlCategory).HasMajorGridlines = False
```

This example turns off all gridlines for all axes in Chart1.

```
For Each a In Charts("Chart1").Axes
    a.HasMajorGridlines = False
    a.HasMinorGridlines = False
Next a
```



# BarGroups Method

On a 2-D chart, returns an object that represents either a single bar chart group (a [ChartGroup](#) object) or a collection of the bar chart groups (a [ChartGroups](#) collection).

*expression*.**BarGroups**(*Index*)

*expression* Required. An expression that returns a **Chart** object.

*Index* Optional **Variant**. Specifies the chart group.

## Example

This example sets the space between bar clusters in the 2-D bar chart group to be 50 percent of the bar width.

```
Charts(1).BarGroups(1).GapWidth = 50
```



# BeginConnect Method

Attaches the beginning of the specified connector to a specified shape. If there's already a connection between the beginning of the connector and another shape, that connection is broken. If the beginning of the connector isn't already positioned at the specified connecting site, this method moves the beginning of the connector to the connecting site and adjusts the size and position of the connector. Use the [EndConnect](#) method to attach the end of the connector to a shape.

*expression*.**BeginConnect**(*ConnectedShape*, *ConnectionSite*)

*expression* Required. An expression that returns a **ConnectorFormat** object.

**ConnectedShape** Required **Shape** object. The shape to attach the beginning of the connector to. The specified **Shape** object must be in the same **Shapes** collection as the connector.

**ConnectionSite** Required **Long**. A connection site on the shape specified by **ConnectedShape**. Must be an integer between 1 and the integer returned by the **ConnectionSiteCount** property of the specified shape. If you want the connector to automatically find the shortest path between the two shapes it connects, specify any valid integer for this argument and then use the [RerouteConnections](#) method after the connector is attached to shapes at both ends.

## Remarks

When you attach a connector to an object, the size and position of the connector are automatically adjusted, if necessary.

## Example

This example adds two rectangles to myDocument and connects them with a curved connector. Notice that the **RerouteConnections** method makes it irrelevant what values you supply for the **ConnectionSite** arguments used with the **BeginConnect** and **EndConnect** methods.

```
Set myDocument = Worksheets(1)
Set s = myDocument.Shapes
Set firstRect = s.AddShape(msoShapeRectangle, 100, 50, 200, 100)
Set secondRect = s.AddShape(msoShapeRectangle, 300, 300, 200, 100)
Set c = s.AddConnector(msoConnectorCurve, 0, 0, 100, 100)
with c.ConnectorFormat
    .BeginConnect ConnectedShape:=firstRect, ConnectionSite:=1
    .EndConnect ConnectedShape:=secondRect, ConnectionSite:=1
    c.RerouteConnections
End With
```



# BeginDisconnect Method

Detaches the beginning of the specified connector from the shape it's attached to. This method doesn't alter the size or position of the connector: the beginning of the connector remains positioned at a connection site but is no longer connected. Use the [EndDisconnect](#) method to detach the end of the connector from a shape.

*expression*.**BeginDisconnect**

*expression* Required. An expression that returns a **ConnectorFormat** object.

## Example

This example adds two rectangles to myDocument, attaches them with a connector, automatically reroutes the connector along the shortest path, and then detaches the connector from the rectangles.

```
Set myDocument = Worksheets(1)
Set s = myDocument.Shapes
Set firstRect = s.AddShape(msoShapeRectangle, 100, 50, 200, 100)
Set secondRect = s.AddShape(msoShapeRectangle, 300, 300, 200, 100)
Set c = s.AddConnector(msoConnectorCurve, 0, 0, 0, 0)
With c.ConnectorFormat
    .BeginConnect firstRect, 1
    .EndConnect secondRect, 1
    c.RerouteConnections
    .BeginDisconnect
    .EndDisconnect
End With
```



[Show All](#)

# BorderAround Method

Adds a border to a range and sets the **Color**, **LineStyle**, and **Weight** properties for the new border. **Variant**.

*expression*.**BorderAround**(*LineStyle*, *Weight*, *ColorIndex*, *Color*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**LineStyle** Optional [XLLineStyle](#). The line style for the border.

XLLineStyle can be one of these XLLineStyle constants.

**xlContinuous** *default*.

**xlDash**

**xlDashDot**

**xlDashDotDot**

**xlDot**

**xlDouble**

**xlLineStyleNone**

**xlSlantDashDot**

**xlLineStyleNone**

**Weight** Optional [XlBorderWeight](#). The border weight.

XlBorderWeight can be one of these XlBorderWeight constants.

**xlHairline**

**xlMedium**

**xlThick**

**xlThin** *default*

**ColorIndex** Optional [XlColorIndex](#). The border color, as an index into the current color palette or as a XlColorIndex constant.

XIColorIndex can be one of these XIColorIndex constants.

**xIColorIndexAutomatic** *default*

**xIColorIndexNone**

**Color** Optional **Variant**. The border color, as an RGB value.

## Remarks

You must specify either *ColorIndex* or *Color*, but not both.

You can specify either *LineStyle* or *Weight*, but not both. If you don't specify either argument, Microsoft Excel uses the default line style and weight.

This method outlines the entire range without filling it in. To set the borders of all the cells, you must set the [Color](#), [LineStyle](#), and [Weight](#) properties for the [Borders](#) collection. To clear the border, you must set the **LineStyle** property to **xlLineStyleNone** for all the cells in the range.

## Example

This example adds a thick red border around the range A1:D4 on Sheet1.

```
Worksheets("Sheet1").Range("A1:D4").BorderAround _  
    ColorIndex:=3, Weight:=xlThick
```



[Show All](#)

# BreakLink Method

Converts formulas linked to other Microsoft Excel sources or OLE sources to values.

*expression*.**BreakLink**(*Name*, *Type*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*Name* Required **String**. The name of the link.

*Type* Required [XLinkType](#). The type of link.

XLinkType can be one of these XLinkType constants.

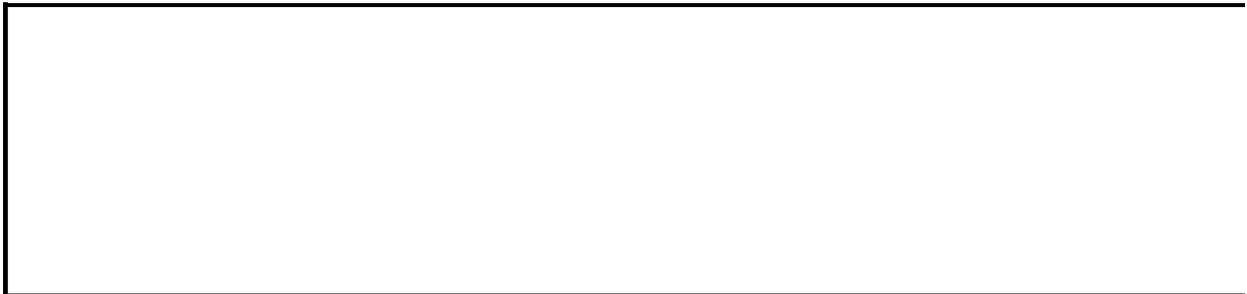
**xLinkTypeExcelLinks** A link to a Microsoft Excel souce.

**xLinkTypeOLELinks** A link to an OLE source.

## Example

In this example, Microsoft Excel converts the first link (an Excel link type) in the active workbook. This example assumes at least one formula exists in the active workbook that links to another Excel source.

```
Sub UseBreakLink()  
  
    Dim astrLinks As Variant  
  
    ' Define variable as an Excel link type.  
    astrLinks = ActiveWorkbook.LinkSources(Type:=xlLinkTypeExcelLink  
  
    ' Break the first link in the active workbook.  
    ActiveWorkbook.BreakLink _  
        Name:=astrLinks(1), _  
        Type:=xlLinkTypeExcelLinks  
  
End Sub
```



# BreakSideBySide Method

Ends side-by-side mode if two windows are in side-by-side mode. Returns a **Boolean** value that represents whether the method was successful.

*expression*.**BreakSideBySide()**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

The following example ends side-by-side mode.

```
Sub CloseSideBySide()  
    ActiveWorkbook.Windows.BreakSideBySide  
End Sub
```



# BringToFront Method

Brings the object to the front of the z-order.

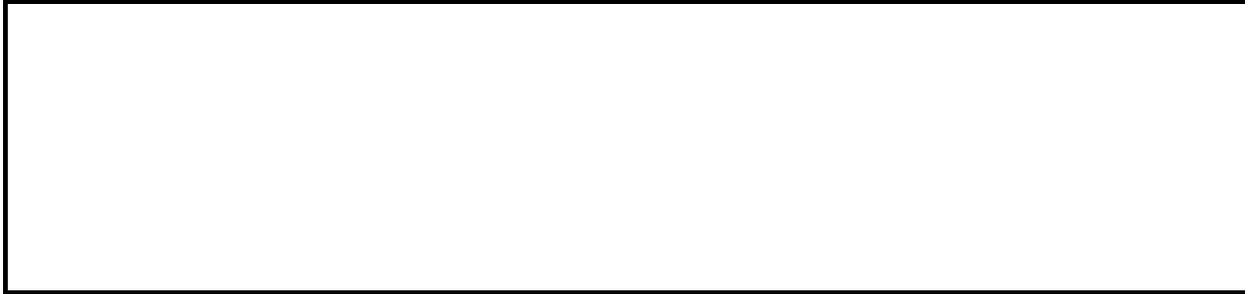
*expression*.**BringToFront**

*expression* Required. An expression that returns an object in the Applies To list.

## Example

This example brings embedded chart one on Sheet1 to the front of the z-order.

```
Worksheets("Sheet1").ChartObjects(1).BringToFront
```



[Show All](#)

# BuildFreeform Method

Builds a freeform object. Returns a [FreeformBuilder](#) object that represents the freeform as it is being built. Use the [AddNodes](#) method to add segments to the freeform. After you have added at least one segment to the freeform, you can use the [ConvertToShape](#) method to convert the **FreeformBuilder** object into a **Shape** object that has the geometric description you've defined in the **FreeformBuilder** object.

*expression*.**BuildFreeform**(*EditingType*, *X1*, *Y1*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*EditingType* Required [MsoEditingType](#). The editing property of the first node.

MsoEditingType can be one of these MsoEditingType constants.

**msoEditingAuto**

**msoEditingCorner**

Cannot be **msoEditingSmooth** or **msoEditingSymmetric**.

**X1** Required **Single**. The position (in points) of the first node in the freeform drawing relative to the upper-left corner of the document.

**Y1** Required **Single**. The position (in points) of the first node in the freeform drawing relative to the upper-left corner of the document.

## Example

This example adds a freeform with five vertices to myDocument.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.BuildFreeform(msoEditingCorner, 360, 200)
    .AddNodes msoSegmentCurve, msoEditingCorner, _
        380, 230, 400, 250, 450, 300
    .AddNodes msoSegmentCurve, msoEditingAuto, 480, 200
    .AddNodes msoSegmentLine, msoEditingAuto, 480, 400
    .AddNodes msoSegmentLine, msoEditingAuto, 360, 200
    .ConvertToShape
End With
```



# Calculate Method

Calculates all open workbooks, a specific worksheet in a workbook, or a specified range of cells on a worksheet, as shown in the following table.

## To calculate

## Follow this example

All open workbooks `Application.Calculate` (or just `Calculate`)

A specific worksheet `Worksheets(1).Calculate`

A specified range `Worksheets(1).Rows(2).Calculate`

*expression*.**Calculate**

*expression* Optional for **Application**, required for **Worksheet** and **Range**. An expression that returns an object in the Applies To list.

## Example

This example calculates the formulas in columns A, B, and C in the used range on Sheet1.

```
Worksheets("Sheet1").UsedRange.Columns("A:C").Calculate
```



# CalculatedFields Method

Returns a [CalculatedFields](#) collection that represents all the calculated fields in the specified PivotTable report. Read-only.

*expression*.**CalculatedFields**

*expression* Required. An expression that returns a **PivotTable** object.

## Example

This example prevents the calculated fields from being dragged to the row position.

```
For Each fld in _  
    Worksheets(1).PivotTables("Pivot1") _  
        .CalculatedFields  
    fld.DragToRow = False  
Next
```



[Show All](#)

# CalculatedItems Method

Returns a [CalculatedItems](#) collection that represents all the calculated items in the specified PivotTable report. Read-only.

*expression*.**CalculatedItems**

*expression* Required. An expression that returns a **PivotField** object.

## Remarks

For [OLAP](#) data sources, this method returns a zero-length collection.

## Example

This example creates a list of calculated items and their formulas.

```
Set pt = Worksheets(1).PivotTables(1)
For Each ci In pt.PivotFields("Sales").CalculatedItems
    r = r + 1
    With Worksheets(2)
        .Cells(r, 1).Value = ci.Name
        .Cells(r, 2).Value = ci.Formula
    End With
Next
```



# CalculateFull Method

Forces a full calculation of the data in all open workbooks.

*expression*.**CalculateFull**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example compares the version of Microsoft Excel with the version of Excel that the workbook was last calculated in. If the two version numbers are different, a full calculation of the data in all open workbooks is performed.

```
If Application.CalculationVersion <> _  
    Workbooks(1).CalculationVersion Then  
    Application.CalculateFull  
End If
```



# CalculateFullRebuild Method

For all open workbooks, forces a full calculation of the data and rebuilds the dependencies.

*expression*.**CalculateFullRebuild**

*expression* Required. An expression that returns one of the objects in the Applies To list.

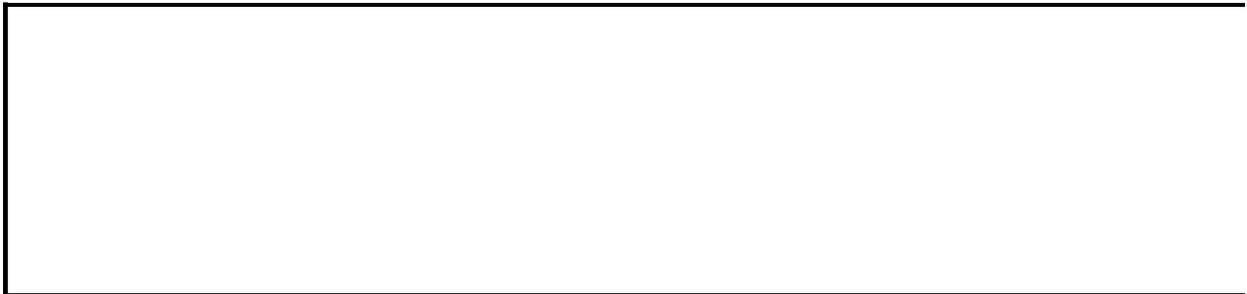
## Remarks

Dependencies are the formulas that depend on other cells. For example, the formula "=A1" depends on cell A1. The **CalculateFullRebuild** method is similar to re-entering all formulas.

## Example

This example compares the version of Microsoft Excel with the version of Excel in which the workbook was last calculated. If the two version numbers are different, a full calculation of the data in all open workbooks is performed and the dependencies are rebuilt.

```
Sub UseCalculateFullRebuild()  
    If Application.CalculationVersion <> _  
        Workbooks(1).CalculationVersion Then  
        Application.CalculateFullRebuild  
    End If  
End Sub
```



# CancelRefresh Method

Cancels all background queries for the specified query table. Use the [Refreshing](#) property to determine whether a background query is currently in progress.

*expression*.**CancelRefresh**

*expression* Required. An expression that returns a **QueryTable** object.

## Example

This example cancels a query table refresh operation.

```
With Worksheets(1).QueryTables(1)  
    If .Refreshing Then .CancelRefresh  
End With
```



# CanCheckIn Method

**True** if Microsoft Excel can check in a specified workbook to a server.  
Read/write **Boolean**.

*expression*.**CanCheckIn**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example checks the server to see if the specified workbook can be checked in. If it can be, it saves and closes the workbook and checks it back into the server.

```
Sub CheckInOut(strWkbCheckIn As String)

    ' Determine if workbook can be checked in.
    If Workbooks(strWkbCheckIn).CanCheckIn = True Then
        Workbooks(strWkbCheckIn).CheckIn
        MsgBox strWkbCheckIn & " has been checked in."
    Else
        MsgBox "This file cannot be checked in " & _
            "at this time. Please try again later."
    End If
End Sub
```



# CanCheckOut Method

**True** if Microsoft Excel can check out a specified workbook from a server.  
Read/write **Boolean**.

*expression*.**CanCheckOut**(*FileName*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**FileName** Required **String**. The name of the file to check out.

## Example

This example verifies that a workbook is not checked out by another user and can be checked out. If the workbook can be checked out, it copies the workbook to the local computer for editing.

```
Sub UseCanCheckOut(docCheckOut As String)

    ' Determine if workbook can be checked out.
    If Workbooks.CanCheckOut(Filename:=docCheckOut) = True Then
        Workbooks.CheckOut (Filename:=docCheckOut)
    Else
        MsgBox "You are unable to check out this document at this ti
    End If

End Sub
```



# CentimetersToPoints Method

Converts a measurement from centimeters to points (one point equals 0.035 centimeters).

*expression*.**CentimetersToPoints**(*Centimeters*)

*expression* Required. An expression that returns an **Application** object.

**Centimeters** Required **Double**. Specifies the centimeter value to be converted to points.

## Example

This example sets the left margin of Sheet1 to 5 centimeters.

```
Worksheets("Sheet1").PageSetup.LeftMargin = _  
    Application.CentimetersToPoints(5)
```



[Show All](#)

# ChangeFileAccess Method

Changes the access permissions for the workbook. This may require an updated version to be loaded from the disk.

*expression*.**ChangeFileAccess**(*Mode*, *WritePassword*, *Notify*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Mode** Required [XlFileAccess](#). Specifies the new access mode.

XlFileAccess can be one of these XlFileAccess constants.

**xlReadWrite**

**xlReadOnly**

**WritePassword** Optional **Variant**. Specifies the write-reserved password if the file is write reserved and **Mode** is **xlReadWrite**. Ignored if there's no password for the file or if **Mode** is **xlReadOnly**.

**Notify** Optional **Variant**. **True** (or omitted) to notify the user if the file cannot be immediately accessed.

## **Remarks**

If you have a file open in read-only mode, you don't have exclusive access to the file. If you change a file from read-only to read/write, Microsoft Excel must load a new copy of the file to ensure that no changes were made while you had the file open as read-only.

## Example

This example sets the active workbook to read-only.

```
ActiveWorkbook.ChangeFileAccess Mode:=xlReadOnly
```



[Show All](#)

# ChangeLink Method

Changes a link from one document to another.

*expression*.**ChangeLink**(*Name*, *NewName*, *Type*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Name** Required **String**. The name of the Microsoft Excel or DDE/OLE link to be changed, as it was returned from the [LinkSources](#) method.

**NewName** Required **String**. The new name of the link.

**Type** Optional [XLLinkType](#). The link type.

XLLinkType can be one of these XLLinkType constants.

**xlLinkTypeExcelLinks** *default*

**xlLinkTypeOLELinks**. Use for both DDE and OLE links.

## Example

This example changes a Microsoft Excel link.

```
ActiveWorkbook.ChangeLink "c:\excel\book1.xls", _  
    "c:\excel\book2.xls", xlExcelLinks
```



# ChangePassword Method

Changes the password for a range that can be edited on a protected worksheet.

*expression*.**ChangePassword**(*Password*)

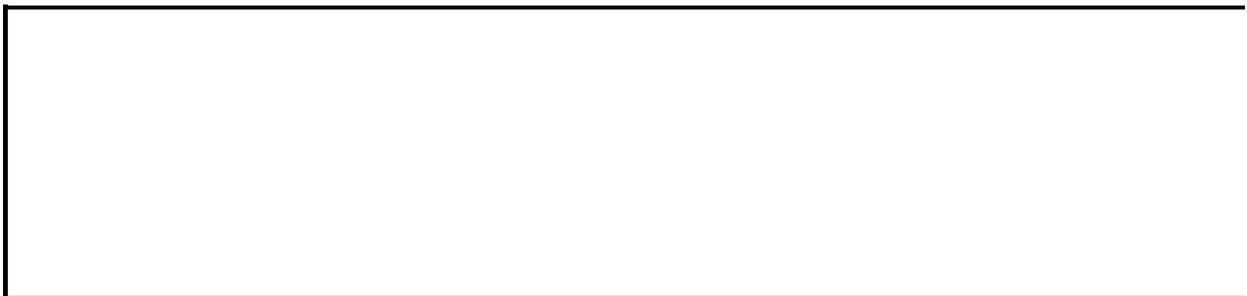
*expression* Required. An expression that returns one of the objects in the Applies To list.

**Password** Required **String**. The new password.

## Example

In this example, Microsoft Excel allows edits to range "A1:A4" on the active worksheet, notifies the user, changes the password for this specified range, and notifies the user of the change. The worksheet must be unprotected before running this code.

```
Sub UseChangePassword()  
  
    Dim wksOne As Worksheet  
    Dim strPassword As String  
  
    Set wksOne = Application.ActiveSheet  
  
    ' Establish a range that can allow edits  
    ' on the protected worksheet.  
  
    strPassword = InputBox("Please enter the password for the range"  
        wksOne.Protection.AllowEditRanges.Add _  
            Title:="Classified", _  
            Range:=Range("A1:A4"), _  
            Password:=strPassword  
  
    strPassword = InputBox("Please enter the new password for the ra  
  
    ' Change the password.  
    wksOne.Protection.AllowEditRanges("Classified").ChangePassword _  
        Password:="strPassword"  
  
    MsgBox "The password for these cells has been changed."  
  
End Sub
```



# ChangeScenario Method

Changes the scenario to have a new set of changing cells and (optionally) scenario values.

*expression*.**ChangeScenario**(*ChangingCells*, *Values*)

*expression* Required. An expression that returns a **Scenario** object.

**ChangingCells** Required **Variant**. A **Range** object that specifies the new set of changing cells for the scenario. The changing cells must be on the same sheet as the scenario.

**Values** Optional **Variant**. An array that contains the new scenario values for the changing cells. If this argument is omitted, the scenario values are assumed to be the current values in the changing cells.

## Remarks

If you specify *Values*, the array must contain an element for each cell in the *ChangingCells* range; otherwise, Microsoft Excel generates an error.

## Example

This example sets the changing cells for scenario one to the range A1:A10 on Sheet1.

```
Worksheets("Sheet1").Scenarios(1).ChangeScenario _  
    Worksheets("Sheet1").Range("A1:A10")
```



# Characters Method

Returns a [Characters](#) object that represents a range of characters within a shape's text frame. You can use the **Characters** object to add and format characters within the text frame.

*expression*.**Characters**(*Start*, *Length*)

*expression* Required. An expression that returns a **Characters** object in the specified text frame.

**Start** Optional **Variant**. The first character to be returned. If this argument is either set to 1 or omitted, the **Characters** method returns a range of characters starting with the first character.

**Length** Optional **Variant**. The number of characters to be returned. If this argument is omitted, the **Characters** method returns the remainder of the string (everything after the character that was set as the **Start** argument).

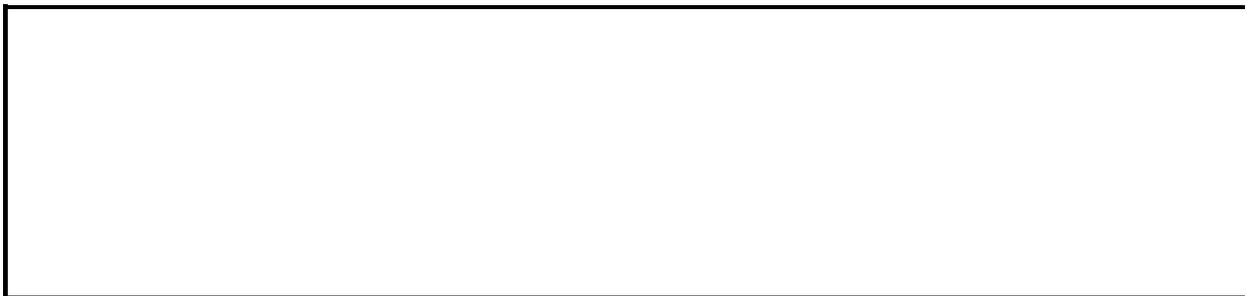
## Remarks

The **Characters** object isn't a collection.

## Example

This example formats as bold the third character in the first shape's text frame on the active worksheet.

```
With ActiveSheet.Shapes(1).TextFrame
    .Characters.Text = "abcdefg"
    .Characters(3, 1).Font.Bold = True
End With
```



# ChartGroups Method

Returns an object that represents either a single chart group (a [ChartGroup](#) object) or a collection of all the chart groups in the chart (a [ChartGroups](#) object). The returned collection includes every type of group.

*expression*.**ChartGroups**(*Index*)

*expression* Required. An expression that returns a **Chart** object.

*Index* Optional **Variant**. The chart group number.

## Example

This example turns on up and down bars for chart group one on Chart1 and then sets their colors. The example should be run on a 2-D line chart containing two series that intersect at one or more data points.

```
With Charts("Chart1").ChartGroups(1)  
    .HasUpDownBars = True  
    .DownBars.Interior.ColorIndex = 3  
    .UpBars.Interior.ColorIndex = 5  
End With
```



# ChartObjects Method

Returns an object that represents either a single embedded chart (a [ChartObject](#) object) or a collection of all the embedded charts (a [ChartObjects](#) object) on the sheet.

*expression*.**ChartObjects**(*Index*)

*expression* Required. An expression that returns an object in the Applies To list. If you specify a **Chart** object, it must be a chart sheet (it cannot be an embedded chart).

**Index** Optional **Variant**. The name or number of the chart. This argument can be an array, to specify more than one chart.

## Remarks

This method isn't equivalent to the [Charts](#) property. This method returns embedded charts; the **Charts** property returns chart sheets. Use the [Chart](#) property to return the **Chart** object for an embedded chart.

## Example

This example adds a title to embedded chart one on Sheet1.

```
With Worksheets("Sheet1").ChartObjects(1).Chart
    .HasTitle = True
    .ChartTitle.Text = "1995 Rainfall Totals by Month"
End With
```

This example creates a new series in embedded chart one on Sheet1. The data source for the new series is the range B1:B10 on Sheet1.

```
Worksheets("Sheet1").ChartObjects(1).Activate
ActiveChart.SeriesCollection.Add _
    source:=Worksheets("Sheet1").Range("B1:B10")
```

This example clears the formatting of embedded chart one on Sheet1.

```
Worksheets("Sheet1").ChartObjects(1).Chart.ChartArea.ClearFormats
```



[Show All](#)

# ChartWizard Method

Modifies the properties of the given chart. You can use this method to quickly format a chart without setting all the individual properties. This method is non-interactive, and it changes only the specified properties.

*expression*.**ChartWizard**(*Source, Gallery, Format, PlotBy, CategoryLabels, SeriesLabels, HasLegend, Title, CategoryTitle, ValueTitle, ExtraTitle*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Source** Optional **Variant**. The range that contains the source data for the new chart. If this argument is omitted, Microsoft Excel edits the active chart sheet or the selected chart on the active worksheet.

**Gallery** Optional [xlChartType](#). The chart type.

xlChartType can be one of these xlChartType constants.

**xlArea**

**xlBar**

**xlColumn**

**xlLine**

**xlPie**

**xlRadar**

**xlXYScatter**

**xlCombination**

**xl3DArea**

**xl3DBar**

**xl3DColumn**

**xl3DLine**

**xl3DPie**

**xl3DSurface**

**xlDoughnut**

**xlDefaultAutoFormat**

**Format** Optional **Variant**. The option number for the built-in autoformats. Can be a number from 1 through 10, depending on the gallery type. If this argument is omitted, Microsoft Excel chooses a default value based on the gallery type and data source.

**PlotBy** Optional **Variant**. Specifies whether the data for each series is in rows or columns. Can be one of the following **XlRowCol** constants: **xlRows** or **xlColumns**.

**CategoryLabels** Optional **Variant**. An integer specifying the number of rows or columns within the source range that contain category labels. Legal values are from 0 (zero) through one less than the maximum number of the corresponding categories or series.

**SeriesLabels** Optional **Variant**. An integer specifying the number of rows or columns within the source range that contain series labels. Legal values are from 0 (zero) through one less than the maximum number of the corresponding categories or series.

**HasLegend** Optional **Variant**. **True** to include a legend.

**Title** Optional **Variant**. The chart title text.

**CategoryTitle** Optional **Variant**. The category axis title text.

**ValueTitle** Optional **Variant**. . The value axis title text

***ExtraTitle*** Optional **Variant**. The series axis title for 3-D charts or the second value axis title for 2-D charts.

## Remarks

If **Source** is omitted and either the selection isn't an embedded chart on the active worksheet or the active sheet isn't an existing chart, this method fails and an error occurs.

## Example

This example reformats Chart1 as a line chart, adds a legend, and adds category and value axis titles.

```
Charts("Chart1").ChartWizard _  
    Gallery:=xlLine, _  
    HasLegend:=True, CategoryTitle:="Year", ValueTitle:="Sales"
```



# CheckAbort Method

Stops recalculation in a Microsoft Excel application.

*expression*.**CheckAbort**(*KeepAbort*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**KeepAbort** Optional **Variant**. Allows recalculation to be performed for a **Range**.

## Example

In this example, Excel stops recalculation in the application, except for cell A10. For you to be able to see the results of this example, other calculations should exist in the application that will allow you to see the differences between the cell designated to continue recalculating and other cells.

```
Sub UseCheckAbort()  
  
    Dim rngSubtotal As Variant  
    Set rngSubtotal = Application.Range("A10")  
  
    ' Stop recalculation except for designated cell.  
    Application.CheckAbort KeepAbort:=rngSubtotal  
  
End Sub
```



# CheckIn Method

Returns a workbook from a local computer to a server, and sets the local workbook to read-only so that it cannot be edited locally. Calling this method will also close the workbook.

*expression*.**CheckIn**(*SaveChanges*, *Comments*, *MakePublic*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**SaveChanges** Optional **Variant**. **True** saves changes and checks in the document. **False** returns the document to a checked-in status without saving revision.

**Comments** Optional **Variant**. Allows the user to enter check-in comments for the revision of the workbook being checked in (applies only if **SaveChanges** equals **True**).

**MakePublic** Optional **Variant**. **True** allows the user to publish the workbook after it has been checked in. This submits the workbook for the approval process, which can eventually result in a version of the workbook being published to users with read-only rights to the workbook (applies only if **SaveChanges** equals **True**).

## Example

This example checks the server to see if the specified workbook can be checked in. If it can, the code saves and closes the workbook and checks it back in to the server.

```
Sub CheckInOut(strWkbCheckIn As String)

    ' Determine if workbook can be checked in.
    If Workbooks(strWkbCheckIn).CanCheckIn = True Then
        Workbooks(strWkbCheckIn).CheckIn
        MsgBox strWkbCheckIn & " has been checked in."
    Else
        MsgBox "This file cannot be checked in " & _
            "at this time. Please try again later."
    End If
End Sub
```



# CheckOut Method

Returns a **String** representing a specified workbook from a server to a local computer for editing.

*expression*.**CheckOut**(*FileName*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**FileName** Required **String**. The name of the file to check out.

## Example

This example verifies that a workbook is not checked out by another user and can be checked out. If the workbook can be checked out, it copies the workbook to the local computer for editing.

```
Sub UseCheckOut(docCheckOut As String)

    ' Determine if workbook can be checked out.
    If Workbooks.CanCheckOut(docCheckOut) = True Then
        Workbooks.CheckOut docCheckOut
    Else
        MsgBox "Unable to check out this document at this time."
    End If

End Sub
```



[Show All](#)

# CheckSpelling Method

 [CheckSpelling method as it applies to the \*\*Application\*\* object.](#)

Checks the spelling of a single word. Returns **True** if the word is found in one of the dictionaries; returns **False** if the word isn't found. **Boolean**.

*expression*.**CheckSpelling**(*Word*, *CustomDictionary*, *IgnoreUppercase*)

*expression* Required. An expression that returns one of the above objects.

**Word** Required **String** (used only with **Application** object). The word you want to check.

**CustomDictionary** Optional **Variant**. A string that indicates the file name of the custom dictionary to be examined if the word isn't found in the main dictionary. If this argument is omitted, the currently specified dictionary is used.

**IgnoreUppercase** Optional **Variant**. **True** to have Microsoft Excel ignore words that are all uppercase. **False** to have Microsoft Excel check words that are all uppercase. If this argument is omitted, the current setting will be used.

 [CheckSpelling method as it applies to the \*\*Range\*\* object.](#)

Checks the spelling of an object. This form has no return value; Microsoft Excel displays the **Spelling** dialog box **Variant**.

*expression*.**CheckSpelling**(*CustomDictionary*, *IgnoreUppercase*, *AlwaysSuggest*, *SpellLang*)

*expression* Required. An expression that returns one of the above objects.

**CustomDictionary** Optional **Variant**. A string that indicates the file name of the custom dictionary to be examined if the word isn't found in the main dictionary. If this argument is omitted, the currently specified dictionary is used.

**IgnoreUppercase** Optional **Variant**. **True** to have Microsoft Excel ignore words

that are all uppercase. **False** to have Microsoft Excel check words that are all uppercase. If this argument is omitted, the current setting will be used.

**AlwaysSuggest** Optional **Variant**. **True** to have Microsoft Excel display a list of suggested alternate spellings when an incorrect spelling is found. **False** to have Microsoft Excel wait for you to input the correct spelling. If this argument is omitted, the current setting will be used.

**SpellLang** Optional **Variant**. The language of the dictionary being used. Can be one of the **MsoLanguageID** values used by the [LanguageID](#) property.

[CheckSpelling method as it applies to the Chart and Worksheet objects.](#)

Checks the spelling of an object. This form has no return value; Microsoft Excel displays the **Spelling** dialog box

*expression*.**CheckSpelling**(*CustomDictionary*, *IgnoreUppercase*, *AlwaysSuggest*, *SpellLang*)

*expression* Required. An expression that returns one of the above objects.

**CustomDictionary** Optional **Variant**. A string that indicates the file name of the custom dictionary to be examined if the word isn't found in the main dictionary. If this argument is omitted, the currently specified dictionary is used.

**IgnoreUppercase** Optional **Variant**. **True** to have Microsoft Excel ignore words that are all uppercase. **False** to have Microsoft Excel check words that are all uppercase. If this argument is omitted, the current setting will be used.

**AlwaysSuggest** Optional **Variant**. **True** to have Microsoft Excel display a list of suggested alternate spellings when an incorrect spelling is found. **False** to have Microsoft Excel wait for you to input the correct spelling. If this argument is omitted, the current setting will be used.

**SpellLang** Optional **Variant**. The language of the dictionary being used. Can be one of the **MsoLanguageID** values used by the [LanguageID](#) property.

## Remarks

To check headers, footers, and objects on a worksheet, use this method on a **Worksheet** object.

To check only cells and notes, use this method with the object returned by the **Cells** method.

## Example

This example checks the spelling on Sheet1.

```
worksheets("Sheet1").CheckSpelling
```



# CircleInvalid Method

Circles invalid entries on the worksheet.

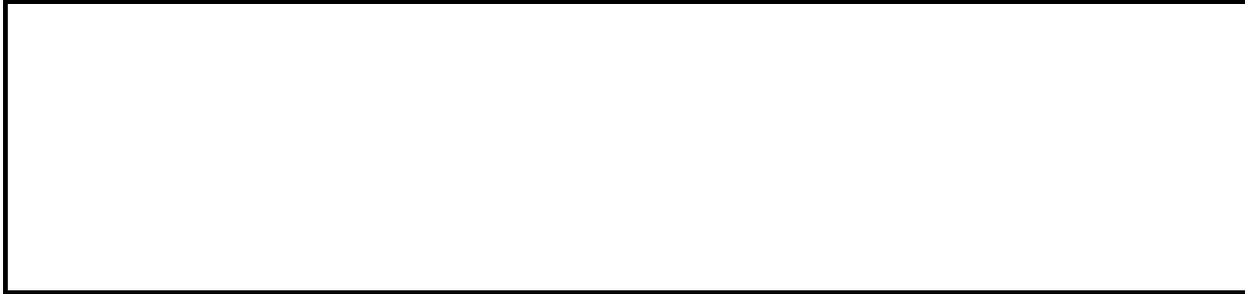
*expression*.**CircleInvalid**

*expression* Required. An expression that returns a **Worksheet** object.

## Example

This example circles invalid entries on worksheet one.

```
worksheets(1).CircleInvalid
```



[Show All](#)

# Clear Method

[Clear method as it applies to the \*\*ChartArea\*\*, \*\*Legend\*\*, and \*\*Range\*\* objects.](#)

Clears the entire object.

*expression*.**Clear**

*expression* Required. An expression that returns one of the above objects.

[Clear method as it applies to the \*\*CellFormat\*\* object.](#)

Clears the criterias set in the **FindFormat** and **ReplaceFormat** properties.

*expression*.**Clear**

*expression* Required. An expression that returns a **CellFormat** object.

[Clear method as it applies to the \*\*XPath\*\* object.](#)

Clears the schema mapping from the cells mapped to the specified XPath.

*expression*.**Clear**

*expression* Required. An expression that returns a **XPath** object.

## Remarks

This method does not clear the data from the cells mapped to the specified XPath. Use the **Clear** method of the **Range** object to clear the data from the cells.

If the specified XPath is mapped in an XML list, then the schema mapping is removed, but the list is not deleted from the worksheet.

## Example

This example clears the formulas and formatting in cells A1:G37 on Sheet1.

```
Worksheets("Sheet1").Range("A1:G37").Clear
```

This example clears the chart area (the chart data and formatting) of Chart1.

```
Charts("Chart1").ChartArea.Clear
```



# ClearArrows Method

Clears the tracer arrows from the worksheet. Tracer arrows are added by using the auditing feature.

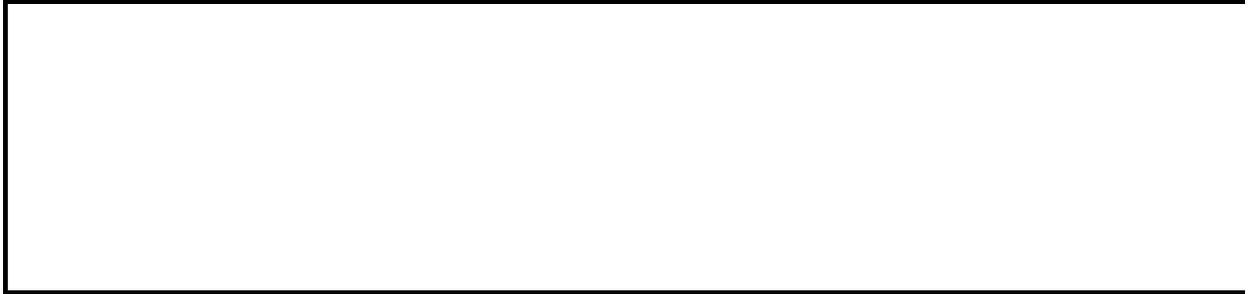
*expression*.**ClearArrows**

*expression* Required. An expression that returns a **Worksheet** object.

## Example

This example clears tracer arrows from Sheet1.

```
worksheets("Sheet1").ClearArrows
```



# ClearCircles Method

Clears circles from invalid entries on the worksheet.

*expression*.**ClearCircles**

*expression* Required. An expression that returns a **Worksheet** object.

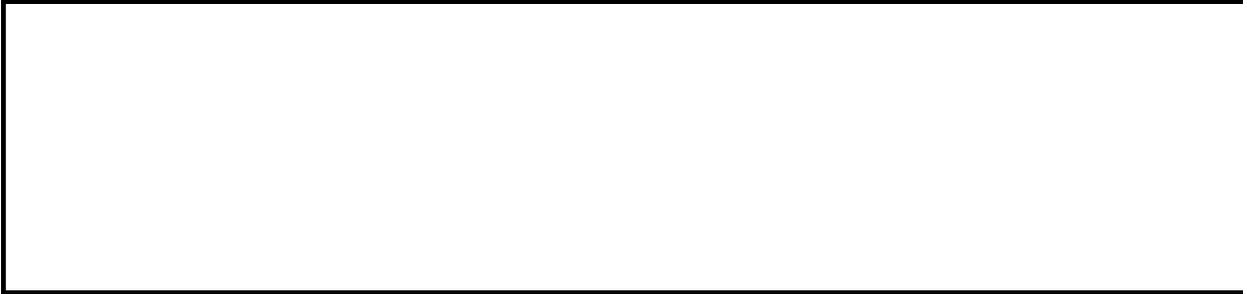
## Remarks

Use the **CircleInvalid** method to circle cells that contain invalid data.

## Example

This example clears circles from invalid entries on worksheet one.

```
worksheets(1).ClearCircles
```



# ClearComments Method

Clears all cell comments from the specified range.

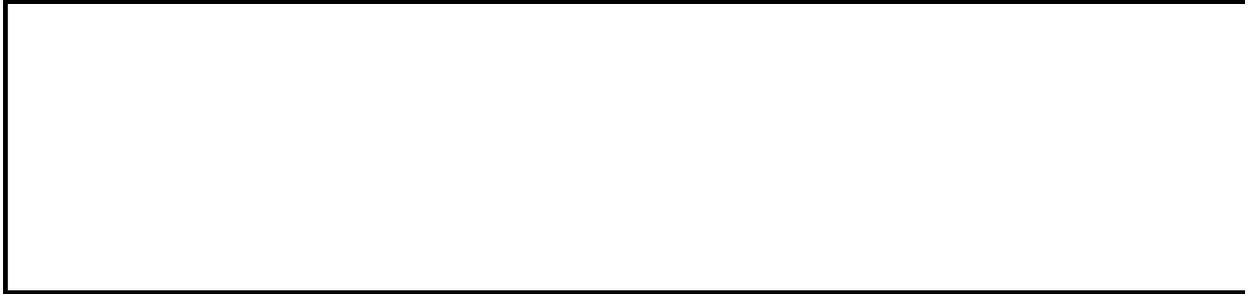
*expression*.**ClearComments**

*expression* Required. An expression that returns a **Range** object.

## Example

This example clears all comments from cell E5.

```
Worksheets(1).Range("e5").ClearComments
```



# ClearContents Method

Clears the formulas from the range. Clears the data from a chart but leaves the formatting.

*expression*.**ClearContents**

*expression* Required. An expression that returns a **ChartArea** or **Range** object.

## Example

This example clears the formulas from cells A1:G37 on Sheet1 but leaves the formatting intact.

```
Worksheets("Sheet1").Range("A1:G37").ClearContents
```

This example clears the chart data from Chart1 but leaves the formatting intact.

```
Charts("Chart1").ChartArea.ClearContents
```



# ClearFormats Method

Clears the formatting of the object.

*expression*.**ClearFormats**

*expression* Required. An expression that returns an object in the Applies To list.

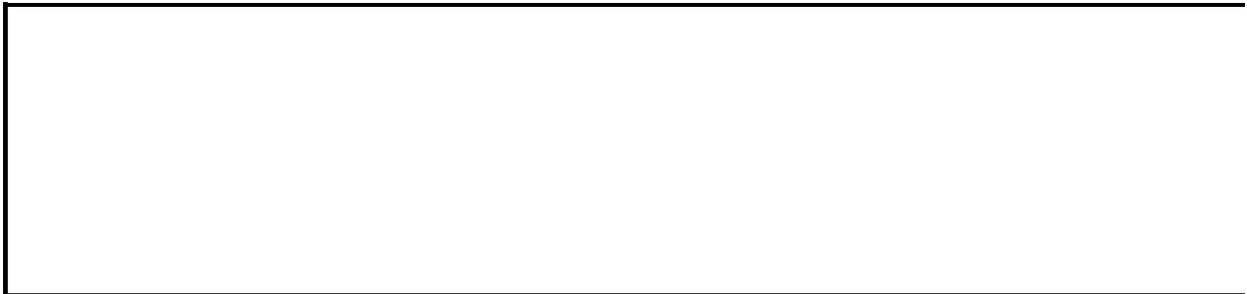
## Example

This example clears all formatting from cells A1:G37 on Sheet1.

```
Worksheets("Sheet1").Range("A1:G37").ClearFormats
```

This example clears the formatting from embedded chart one on Sheet1.

```
Worksheets("Sheet1").ChartObjects(1).Chart.ChartArea.ClearFormats
```



# ClearNotes Method

Clears notes and sound notes from all the cells in the specified range.

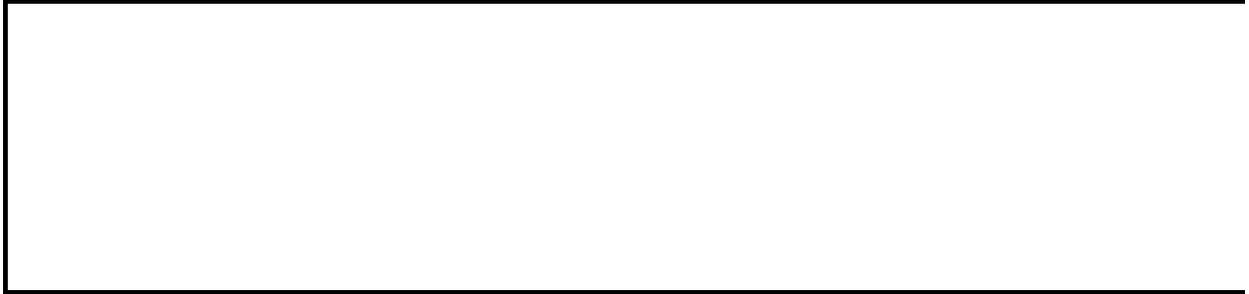
*expression*.**ClearNotes**

*expression* Required. An expression that returns a **Range** object.

## Example

This example clears all notes and sound notes from columns A through C on Sheet1.

```
Worksheets("Sheet1").Columns("A:C").ClearNotes
```



# ClearOutline Method

Clears the outline for the specified range.

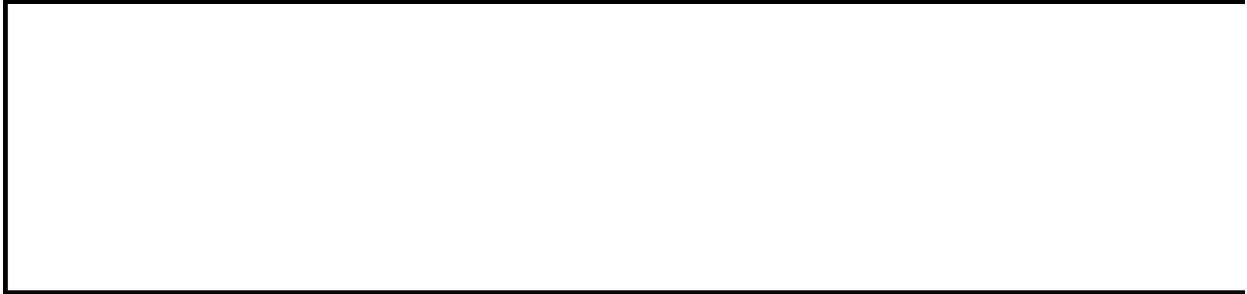
*expression*.**ClearOutline**

*expression* Required. An expression that returns a **Range** object.

## Example

This example clears the outline for the range A1:G37 on Sheet1.

```
Worksheets("Sheet1").Range("A1:G37").ClearOutline
```



# ClearSettings Method

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Removes the specified data binding.

*expression*.**ClearSettings**

*expression* Required. An expression that returns an [XmlDataBinding](#) object.



[Show All](#)

# CloneNode Method

Clones a diagram node. Returns a **DiagramNode** object representing the cloned node.

*expression.CloneNode(copyChildren, pTargetNode, pos)*

*expression* Required. An expression that returns one of the objects in the Applies To list.

*copyChildren* Required **Boolean**. **True** to clone the diagram nodes children as well.

*pTargetNode* Required **DiagramNode** object. An expression that returns a **DiagramNode** object that represents the node where the new node will be placed.

*pos* Optional [MsoRelativeNodePosition](#). If *pTargetNode* is specified, indicates where the node will be added relative to *pTargetNode*.

MsoRelativeNodePosition can be one of these MsoRelativeNodePosition constants.

**msoAfterLastSibling**

**msoAfterNode** *default*

**msoBeforeFirstSibling**

**msoBeforeNode**

## Example

The following example creates a diagram and clones the newest-created node.

```
Sub CloneANode()  
  
    Dim nodRoot As DiagramNode  
    Dim shpDiagram As Shape  
    Dim nodFourthNode As DiagramNode  
    Dim nodDuplicate As DiagramNode  
    Dim intCount As Integer  
  
    Set shpDiagram = ActiveSheet.Shapes.AddDiagram( _  
        Type:=msoDiagramOrgChart, Left:=10, _  
        Top:=15, Width:=400, Height:=475)  
    Set nodRoot = shpDiagram.DiagramNode.Children.AddNode  
  
    ' Add subordinate nodes to the root node  
    For intCount = 1 To 4  
        nodRoot.Children.AddNode  
    Next  
  
    Set nodFourthNode = nodRoot.Children.Item(4)  
  
    'Clone the most recently created child node  
    Set nodDuplicate = nodRoot.Children.Item(1).CloneNode(copyChildr  
        pTargetNode:=nodFourthNode, pos:=msoAfterNode)  
  
End Sub
```



[Show All](#)

# Close Method

 [Close method as it applies to the \*\*Window\*\* object.](#)

Closes the object. **Boolean**.

*expression*.**Close**(*SaveChanges*, *Filename*, *RouteWorkbook*)

*expression* Required. An expression that returns one of the above objects.

**SaveChanges** Optional **Variant**. If there are no changes to the workbook, this argument is ignored. If there are changes to the workbook and the workbook appears in other open windows, this argument is ignored. If there are changes to the workbook but the workbook doesn't appear in any other open windows, this argument specifies whether changes should be saved, as shown in the following table.

Value	Action
<b>True</b>	Saves the changes to the workbook. If there is not yet a file name associated with the workbook, then <b>FileName</b> is used. If <b>FileName</b> is omitted, the user is asked to supply a file name.
<b>False</b>	Does not save the changes to this file.
Omitted	Displays a dialog box asking the user whether or not to save changes.

**FileName** Optional **Variant**. Save changes under this file name.

**RouteWorkbook** Optional **Variant**. If the workbook doesn't need to be routed to the next recipient (if it has no routing slip or has already been routed), this argument is ignored. Otherwise, Microsoft Excel routes the workbook as shown in the following table.

Value	Meaning
<b>True</b>	Sends the workbook to the next recipient.
<b>False</b>	Doesn't send the workbook.
Omitted	Displays a dialog box asking the user whether the workbook should be

sent.

[Close method as it applies to the \*\*Workbooks\*\* object.](#)

Closes the object.

*expression*.**Close**

*expression* Required. An expression that returns one of the above objects.

[Close method as it applies to the \*\*Workbook\*\* object.](#)

Closes the object.

*expression*.**Close**(*SaveChanges*, *Filename*, *RouteWorkbook*)

*expression* Required. An expression that returns one of the above objects.

**SaveChanges** Optional **Variant**. If there are no changes to the workbook, this argument is ignored. If there are changes to the workbook and the workbook appears in other open windows, this argument is ignored. If there are changes to the workbook but the workbook doesn't appear in any other open windows, this argument specifies whether changes should be saved, as shown in the following table.

<b>Value</b>	<b>Action</b>
<b>True</b>	Saves the changes to the workbook. If there is not yet a file name associated with the workbook, then <b>FileName</b> is used. If <b>FileName</b> is omitted, the user is asked to supply a file name.
<b>False</b>	Does not save the changes to this file.
Omitted	Displays a dialog box asking the user whether or not to save changes.

**FileName** Optional **Variant**. Save changes under this file name.

**RouteWorkbook** Optional **Variant**. If the workbook doesn't need to be routed to the next recipient (if it has no routing slip or has already been routed), this argument is ignored. Otherwise, Microsoft Excel routes the workbook as shown in the following table.

<b>Value</b>	<b>Meaning</b>
<b>True</b>	Sends the workbook to the next recipient.
<b>False</b>	Doesn't send the workbook.
Omitted	Displays a dialog box asking the user whether the workbook should be sent.

## Remarks

Closing a workbook from Visual Basic doesn't run any Auto\_Close macros in the workbook. Use the [RunAutoMacros](#) method to run the auto close macros.

## Example

This example closes Book1.xls and discards any changes that have been made to it.

```
Workbooks("BOOK1.XLS").Close SaveChanges:=False
```

This example closes all open workbooks. If there are changes in any open workbook, Microsoft Excel displays the appropriate prompts and dialog boxes for saving changes.

```
Workbooks.Close
```



# ColumnDifferences Method

Returns a [Range](#) object that represents all the cells whose contents are different from the comparison cell in each column.

*expression*.**ColumnDifferences**(*Comparison*)

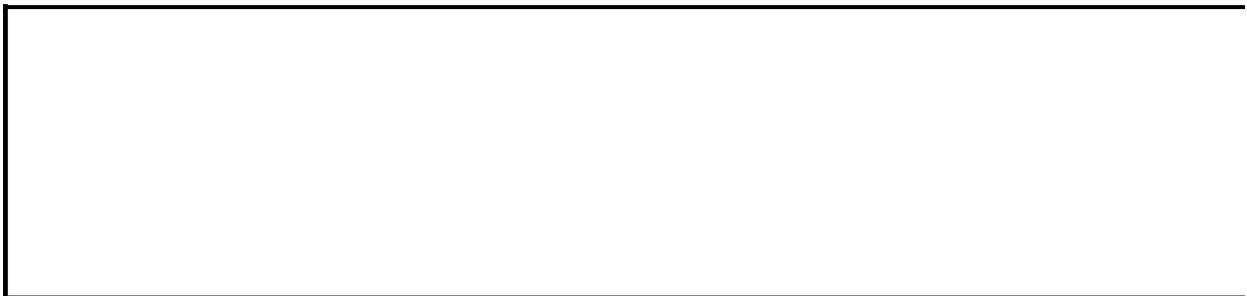
*expression* Required. An expression that returns a **Range** object containing the cells to compare.

**Comparison** Required **Variant**. A single cell to compare to the specified range.

## Example

This example selects the cells in column A on Sheet1 whose contents are different from cell A4.

```
Worksheets("Sheet1").Activate  
Set r1 = ActiveSheet.Columns("A").ColumnDifferences( _  
    Comparison:=ActiveSheet.Range("A4"))  
r1.Select
```



# ColumnGroups Method

On a 2-D chart, returns an object that represents either a single column chart group (a [ChartGroup](#) object) or a collection of the column chart groups (a [ChartGroups](#) collection).

*expression*.**ColumnGroups**(*Index*)

*expression* Required. An expression that returns a **Chart** object.

*Index* Optional **Variant**. Specifies the chart group.

## Example

This example sets the space between column clusters in the 2-D column chart group to be 50 percent of the column width.

```
Charts(1).ColumnGroups(1).GapWidth = 50
```



# CompareSideBySideWith Method

Opens two windows in side-by-side mode. Returns a **Boolean** value.

*expression*.**CompareSideBySideWith**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

You cannot use the **CompareSideBySideWith** method with the [Application](#) object or the [ActiveWorkbook](#) property.

---

---

---

# ConnectData Method

Adds new topics from a real-time data server. The **ConnectData** method is called when a file is opened that contains real-time data functions or when a user types in a new formula which contains the RTD function.

*expression*.**ConnectData**(*TopicID*, *Strings*, *GetNewValues*)

*expression* Required. An expression that returns an **IRtdServer** object.

**TopicID** Required **Long**. A unique value, assigned by Microsoft Excel, which identifies the topic.

**Strings** Required **VARIANT**. A single-dimensional array of strings identifying the topic.

**GetNewValues** Required **Boolean**. **True** to determine if new values are to be acquired.



[Show All](#)

# Consolidate Method

Consolidates data from multiple ranges on multiple worksheets into a single range on a single worksheet. **Variant**.

*expression*.**Consolidate**(*Sources*, *Function*, *TopRow*, *LeftColumn*, *CreateLinks*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Sources** Optional **Variant**. The sources of the consolidation as an array of text reference strings in R1C1-style notation. The references must include the full path of sheets to be consolidated.

**Function** Optional [XlConsolidationFunction](#).

XlConsolidationFunction can be one of these XlConsolidationFunction constants.

**xlAverage** *default*.

**xlCount**

**xlCountNums**

**xlMax**

**xlMin**

**xlProduct**

**xlStDev**

**xlStDevP**

**xlSum**

**xlVar**

## **xlVarP**

**TopRow** Optional **Variant**. **True** to consolidate data based on column titles in the top row of the consolidation ranges. **False** to consolidate data by position. The default value is **False**.

**LeftColumn** Optional **Variant**. **True** to consolidate data based on row titles in the left column of the consolidation ranges. **False** to consolidate data by position. The default value is **False**.

**CreateLinks** Optional **Variant**. **True** to have the consolidation use worksheet links. **False** to have the consolidation copy the data. The default value is **False**.

## Example

This example consolidates data from Sheet2 and Sheet3 onto Sheet1, using the SUM function.

```
Worksheets("Sheet1").Range("A1").Consolidate _  
    Sources:=Array("Sheet2!R1C1:R37C6", "Sheet3!R1C1:R37C6"), _  
    Function:=xlSum
```



[Show All](#)

# Convert Method

Converts the current diagram to a different diagram.

*expression*.**Convert**(*Type*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*Type* Required [MsoDiagramType](#). The type of diagram to convert to.

MsoDiagramType can be one of these MsoDiagramType constants.

**msoDiagramCycle** A process diagram with a continuous cycle diagram type.

**msoDiagramMixed** A mixed diagram type.

**msoDiagramOrgChart** A hierarchical relationship diagram type.

**msoDiagramPyramid** A foundation based relationships diagram type.

**msoDiagramRadial** A diagram type showing relationships of a core element.

**msoDiagramTarget** A diagram type showing steps toward a goal.

**msoDiagramVenn** A diagram type showing areas of overlap between elements.

## Example

This example adds a radial diagram to the active worksheet and then converts it to a target diagram.

```
Sub ConvertDiagram()  
  
    Dim wksSheet As Worksheet  
    Dim shDiagram As Shape  
  
    Set wksSheet = ActiveSheet  
    Set shDiagram = wksSheet.Shapes.AddDiagram( _  
        Type:=msoDiagramRadial, _  
        Left:=20, Top:=40, _  
        Width:=400, Height:=200)  
  
    ' Fill the diagram to make it visible to the user  
    shDiagram.Fill.Visible = msoTrue  
  
    ' Convert the diagram.  
    shDiagram.Diagram.Convert Type:=msoDiagramTarget  
  
End Sub
```



[Show All](#)

# ConvertFormula Method

Converts cell references in a formula between the A1 and R1C1 reference styles, between relative and absolute references, or both. **Variant**.

*expression*.**ConvertFormula**(*Formula*, *FromReferenceStyle*, *ToReferenceStyle*, *ToAbsolute*, *RelativeTo*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Formula** Required **Variant**. A string that contains the formula you want to convert. This must be a valid formula, and it must begin with an equal sign.

**FromReferenceStyle** Required [XlReferenceStyle](#). The reference style of the formula.

XlReferenceStyle can be one of these XlReferenceStyle constants.

**xlA1**

**xlR1C1**

**ToReferenceStyle** Optional [XlReferenceStyle](#). The reference style you want returned. If this argument is omitted, the reference style isn't changed; the formula stays in the style specified by **FromReferenceStyle**.

XlReferenceStyle can be one of these XlReferenceStyle constants.

**xlA1**

**xlR1C1**

**ToAbsolute** Optional [XlReferenceStyle](#). Specifies the converted reference type. If this argument is omitted, the reference type isn't changed.

XlReferenceStyle can be one of these XlReferenceStyle constants.

**xlAbsolute**

**xlAbsRowRelColumn**

**xlRelRowAbsColumn**

**xlRelative**

***RelativeTo*** Optional **Variant**. Optional **Variant**. A **Range** object that contains one cell. Relative references relate to this cell.

## Example

This example converts a SUM formula that contains R1C1-style references to an equivalent formula that contains A1-style references, and then it displays the result.

```
inputFormula = "=SUM(R10C2:R15C2)"  
MsgBox Application.ConvertFormula( _  
    formula:=inputFormula, _  
    fromReferenceStyle:=xlR1C1, _  
    toReferenceStyle:=xlA1)
```



# ConvertToShape Method

Creates a shape that has the geometric characteristics of the specified **FreeformBuilder** object. Returns a [Shape](#) object that represents the new shape.

**Note** You must apply the [AddNodes](#) method to a **FreeformBuilder** object at least once before you use the **ConvertToShape** method.

*expression*.**ConvertToShape**

*expression* Required. An expression that returns a **FreeformBuilder** object.

## Example

This example adds a freeform with five vertices to myDocument.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.BuildFreeform(msoEditingCorner, 360, 200)
    .AddNodes msoSegmentCurve, msoEditingCorner, _
        380, 230, 400, 250, 450, 300
    .AddNodes msoSegmentCurve, msoEditingAuto, 480, 200
    .AddNodes msoSegmentLine, msoEditingAuto, 480, 400
    .AddNodes msoSegmentLine, msoEditingAuto, 360, 200
    .ConvertToShape
End With
```



[Show All](#)

# Copy Method

 [Copy method as it applies to the \*\*Range\*\* object.](#)

Copies the range to the specified range or to the Clipboard.

*expression*.**Copy**(*Destination*)

*expression* Required. An expression that returns a **Range** object.

**Destination** Optional **Variant**. Specifies the new range to which the specified range will be copied. If this argument is omitted, Microsoft Excel copies the range to the Clipboard.

 [Copy method as it applies to the \*\*ChartArea\*\*, \*\*ChartObject\*\*, \*\*ChartObjects\*\*, \*\*OLEObject\*\*, \*\*OLEObjects\*\*, \*\*Point\*\*, and \*\*Series\*\* objects.](#)

Copies the object to the Clipboard. Copies a picture of the point or series to the Clipboard.

*expression*.**Copy**

*expression* Required. An expression that returns one of the above objects.

 [Copy method as it applies to the \*\*Chart\*\*, \*\*Charts\*\*, \*\*Sheets\*\*, \*\*Worksheet\*\*, and \*\*Worksheets\*\* objects.](#)

Copies the sheet to another location in the workbook.

*expression*.**Copy**(*Before*, *After*)

*expression* Required. An expression that returns one of the above objects.

**Before** Optional **Variant**. The sheet before which the copied sheet will be placed. You cannot specify **Before** if you specify **After**.

**After** Optional **Variant**. The sheet after which the copied sheet will be placed.

You cannot specify *After* if you specify *Before*.

## Remarks

If you don't specify either **Before** or **After**, Microsoft Excel creates a new workbook that contains the copied sheet.

 [Copy method as it applies to the \*\*Shape\*\* object.](#)

Copies the object to the Clipboard.

*expression*.**Copy**

*expression* Required. An expression that returns a **Shape** object.

## Example

 [Copy method as it applies to the \*\*Range\*\* object.](#)

This example copies the formulas in cells A1:D4 on Sheet1 into cells E5:H8 on Sheet2.

```
Worksheets("Sheet1").Range("A1:D4").Copy _  
    destination:=Worksheets("Sheet2").Range("E5")
```

 [Copy method as it applies to the \*\*Chart, Charts, Sheets, Worksheet, and Worksheets\*\* objects.](#)

This example copies Sheet1, placing the copy after Sheet3.

```
Worksheets("Sheet1").Copy After:=Worksheets("Sheet3")
```



# CopyFromRecordset Method

Copies the contents of an ADO or DAO **Recordset** object onto a worksheet, beginning at the upper-left corner of the specified range. If the **Recordset** object contains fields with OLE objects in them, this method fails.

*expression*.**CopyFromRecordset**(*Data*, *MaxRows*, *MaxColumns*)

*expression* Required. An expression that returns a **Range** object.

**Data** Required **Variant**. The **Recordset** object to copy into the range.

**MaxRows** Optional **Variant**. The maximum number of records to copy onto the worksheet. If this argument is omitted, all the records in the **Recordset** object are copied.

**MaxColumns** Optional **Variant**. The maximum number of fields to copy onto the worksheet. If this argument is omitted, all the fields in the **Recordset** object are copied.

## Remarks

Copying begins at the current row of the **Recordset** object. After copying is completed, the **EOF** property of the **Recordset** object is **True**.

## Example

This example copies the field names from a DAO **Recordset** object into the first row of a worksheet and formats the names as bold. The example then copies the recordset onto the worksheet, beginning at cell A2.

```
For iCols = 0 to rs.Fields.Count - 1
    ws.Cells(1, iCols + 1).Value = rs.Fields(iCols).Name
Next
ws.Range(ws.Cells(1, 1), _
    ws.Cells(1, rs.Fields.Count)).Font.Bold = True
ws.Range("A2").CopyFromRecordset rs
```



[Show All](#)

# CopyPicture Method

 [CopyPicture method as it applies to the \*\*Range\*\* object.](#)

Copies the selected object to the Clipboard as a picture. **Variant.**

*expression*.**CopyPicture**(*Appearance*, *Format*)

*expression* Required. An expression that returns one of the above objects.

**Appearance** Optional [XIPictureAppearance](#). Specifies how the picture should be copied.

XIPictureAppearance can be one of these XIPictureAppearance constants.

**xlPrinter**. The picture is copied as it will look when it's printed.

**xlScreen** *default*. The picture is copied to resemble its display on the screen as closely as possible

**Format** Optional [XICopyPictureFormat](#). The format of the picture.

XICopyPictureFormat can be one of these XICopyPictureFormat constants.

**xlBitmap**

**xlPicture** *default*

 [CopyPicture method as it applies to the \*\*ChartObject\*\*, \*\*ChartObjects\*\*, \*\*OLEObject\*\*, and \*\*OLEObjects\*\* objects.](#)

Copies the selected object to the Clipboard as a picture. **Variant.**

*expression*.**CopyPicture**(*Appearance*, *Format*)>

*expression* Required. An expression that returns one of the above objects.

**Appearance** Optional [XIPictureAppearance](#). Specifies how the picture should be copied.

XlPictureAppearance can be one of these XlPictureAppearance constants.  
**xlPrinter**. The picture is copied as it will look when it's printed.  
**xlScreen** *default*. The picture is copied to resemble its display on the screen as closely as possible

**Format** Optional [XlCopyPictureFormat](#). The format of the picture.

XlCopyPictureFormat can be one of these XlCopyPictureFormat constants.

**xlBitmap**

**xlPicture** *default*

 [CopyPicture](#) method as it applies to the **Chart** object.

Copies the selected object to the Clipboard as a picture.

*expression*.**CopyPicture**(*Appearance*, *Format*, *Size*)

*expression* Required. An expression that returns one of the above objects.

**Appearance** Optional [XlPictureAppearance](#). Specifies how the picture should be copied.

XlPictureAppearance can be one of these XlPictureAppearance constants.  
**xlPrinter**. The picture is copied as it will look when it's printed.  
**xlScreen** *default*. The picture is copied to resemble its display on the screen as closely as possible

**Format** Optional [XlCopyPictureFormat](#). The format of the picture.

XlCopyPictureFormat can be one of these XlCopyPictureFormat constants.

**xlBitmap**

**xlPicture** *default*

**Size** Optional [XlPictureAppearance](#). The size of the copied picture when the object is a chart on a chart sheet (not embedded on a worksheet).

XlPictureAppearance can be one of these XlPictureAppearance constants.

**xlPrinter** *default*. The picture is copied to match its printed size as closely as possible.

**xlScreen**. The picture is copied to match the size of its display on the screen as closely as possible.

 [CopyPicture method as it applies to the \*\*Shape\*\* object.](#)

Copies the selected object to the Clipboard as a picture.

*expression*.**CopyPicture**(*Appearance*, *Format*)

*expression* Required. An expression that returns one of the above objects.

**Appearance** Optional [XlPictureAppearance](#). Specifies how the picture should be copied.

XlPictureAppearance can be one of these XlPictureAppearance constants.

**xlPrinter**. The picture is copied as it will look when it's printed.

**xlScreen** *default*. The picture is copied to resemble its display on the screen as closely as possible

**Format** Optional [XlCopyPictureFormat](#). The format of the picture.

XlCopyPictureFormat can be one of these XlCopyPictureFormat constants.

**xlBitmap**

**xlPicture** *default*

## **Remarks**

If you copy a range, it must be made up of adjacent cells.

## Example

This example copies a screen image of cells A1:D4 on Sheet1 to the Clipboard, and then it pastes the bitmap to another location on Sheet1.

```
Worksheets("Sheet1").Range("A1:D4").CopyPicture xlScreen, xlBitmap  
Worksheets("Sheet1").Paste _  
    Destination:=Worksheets("Sheet1").Range("E6")
```



[Show All](#)

# CreateCubeFile Method

Creates a cube file from a PivotTable report connected to an [Online Analytical Processing \(OLAP\)](#) data source.

*expression*.**CreateCubeFile**(*File*, *Measures*, *Levels*, *Members*, *Properties*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**File** Required **String**. The name of the cube file to be created. It will overwrite the file if it already exists.

**Measures** Optional **VARIANT**. An array of unique names of measures that are to be part of the slice.

**Levels** Optional **VARIANT**. An array of strings. Each array item is a unique level name. It represents the lowest level of a hierarchy that is in the slice.

**Members** Optional **VARIANT**. An array of string arrays. The elements correspond, in order, to the hierarchies represented in the **Levels** array. Each element is an array of string arrays that consists of the unique names of the top level members in the dimension that are to be included in the slice.

**Properties** Optional **Boolean**. **False** results in no member properties being included in the slice. The default value is **True**.

## Example

This example creates a cube file titled "CustomCubeFile" on drive C:\ with no member properties to be included in the slice. With the *Measures*, *Levels*, and *Members* arguments omitted from this example, the cube file will end up matching the view of the PivotTable report. This example assumes a PivotTable report connected to an OLAP data source exists on the active worksheet.

```
Sub UseCreateCubeFile()  
    ActiveSheet.PivotTables(1).CreateCubeFile _  
        File:="C:\CustomCubeFile", Properties:=False  
End Sub
```



# CreateNames Method

Creates names in the specified range, based on text labels in the sheet.

*expression*.CreateNames(**Top**, **Left**, **Bottom**, **Right**)

*expression* Required. An expression that returns a **Range** object.

**Top** Optional **Variant**. **True** to create names by using labels in the top row. The default value is **False**.

**Left** Optional **Variant**. **True** to create names by using labels in the left column. The default value is **False**.

**Bottom** Optional **Variant**. **True** to create names by using labels in the bottom row. The default value is **False**.

**Right** Optional **Variant**. **True** to create names by using labels in the right column. The default value is **False**.

## Remarks

If you don't specify one of *Top*, *Left*, *Bottom*, or *Right*, Microsoft Excel guesses the location of the text labels, based on the shape of the specified range.

## Example

This example creates names for cells B1:B3 based on the text in cells A1:A3. Note that you must include the cells that contain the names in the range, even though the names are created only for cells B1:B3.

```
Set rangeToName = Worksheets("Sheet1").Range("A1:B3")  
rangeToName.CreateNames Left:=True
```



# CreateNewDocument Method

Creates a new document linked to the specified hyperlink.

*expression*.CreateNewDocument(*Filename*, *EditNow*, *Overwrite*)

*expression* An expression that returns a **Hyperlink** object.

**Filename** Required **String**. The file name of the specified document.

**EditNow** Required **Boolean**. **True** to have the specified document open immediately in its associated editing environment.. The default value is **True**.

**Overwrite** Required **Boolean**. **True** to overwrite any existing file of the same name in the same folder. **False** if any existing file of the same name is preserved and the **Filename** argument specifies a new file name. The default value is **False**.

## Example

This example creates a new document based on the new hyperlink in the first worksheet and then loads the document into Microsoft Excel for editing. The document is called "Report.xls," and it overwrites any file of the same name in the \\Server1\Annual folder.

```
With Worksheets(1)
    Set objHyper = _
        .Hyperlinks.Add(Anchor:=.Range("A10"), _
            Address:"\\Server1\Annual\Report.xls")
    objHyper.CreateNewDocument _
        FileName:"\\Server1\Annual\Report.xls", _
        EditNow:=True, Overwrite:=True
End With
```



[Show All](#)

# CreatePivotTable Method

Creates a PivotTable report based on a [PivotCache](#) object. Returns a [PivotTable](#) object.

*expression*.CreatePivotTable(*TableDestination*, *TableName*, *ReadData*, *DefaultVersion*)

*expression* An expression that returns a **PivotCache** object.

**TableDestination** Required **Variant**. The cell in the upper-left corner of the PivotTable report's destination range (the range on the worksheet where the resulting PivotTable report will be placed). The destination range must be on a worksheet in the workbook that contains the **PivotCache** object specified by *expression* .

**TableName** Optional **Variant**. The name of the new PivotTable report.

**ReadData** Optional **Variant**. **True** to create a PivotTable cache that contains all of the records from the external database; this cache can be very large. **False** to enable setting some of the fields as server-based page fields before the data is actually read.

**DefaultVersion** Optional **Variant**. The default version of the PivotTable report.

## Remarks

For an alternative way to create a PivotTable report based on a PivotTable cache, see the [Add](#) method of the **PivotTable** object.

## Example

This example creates a new PivotTable cache based on an [OLAP provider](#), and then it creates a new PivotTable report based on the cache, at cell A3 on the active worksheet.

```
With ActiveWorkbook.PivotCaches.Add(SourceType:=xlExternal)
    .Connection = _
        "OLEDB;Provider=MSOLAP;Location=srvdata;Initial Catalog=Nati
    .CommandType = xlCmdCube
    .CommandText = Array("Sales")
    .MaintainConnection = True
    .CreatePivotTable TableDestination:=Range("A3"), _
        TableName:= "PivotTable1"
End With
With ActiveSheet.PivotTables("PivotTable1")
    .SmallGrid = False
    .PivotCache.RefreshPeriod = 0
    With .CubeFields("[state]")
        .Orientation = xlColumnField
        .Position = 1
    End With
    With .CubeFields("[Measures].[Count Of au_id]")
        .Orientation = xlDataField
        .Position = 1
    End With
End With
```

This example creates a new PivotTable cache using an ADO connection to Microsoft Jet, and then it creates a new PivotTable report based on the cache, at cell A3 on the active worksheet.

```
Dim cnnConn As ADODB.Connection
Dim rstRecordset As ADODB.Recordset
Dim cmdCommand As ADODB.Command

' Open the connection.
Set cnnConn = New ADODB.Connection
With cnnConn
    .ConnectionString = _
        "Provider=Microsoft.Jet.OLEDB.4.0"
    .Open "C:\perfdate\record.mdb"
End With
```

```

' Set the command text.
Set cmdCommand = New ADODB.Command
Set cmdCommand.ActiveConnection = cnnConn
With cmdCommand
    .CommandText = "Select Speed, Pressure, Time From DynoRun"
    .CommandType = adCmdText
    .Execute
End With

' Open the recordset.
Set rstRecordset = New ADODB.Recordset
Set rstRecordset.ActiveConnection = cnnConn
rstRecordset.Open cmdCommand

' Create a PivotTable cache and report.
Set objPivotCache = ActiveWorkbook.PivotCaches.Add( _
    SourceType:=xlExternal)
Set objPivotCache.Recordset = rstRecordset
With objPivotCache
    .CreatePivotTable TableDestination:=Range("A3"), _
        TableName:="Performance"
End With

With ActiveSheet.PivotTables("Performance")
    .SmallGrid = False
    With .PivotFields("Pressure")
        .Orientation = xlRowField
        .Position = 1
    End With
    With .PivotFields("Speed")
        .Orientation = xlColumnField
        .Position = 1
    End With
    With .PivotFields("Time")
        .Orientation = xlDataField
        .Position = 1
    End With
End With

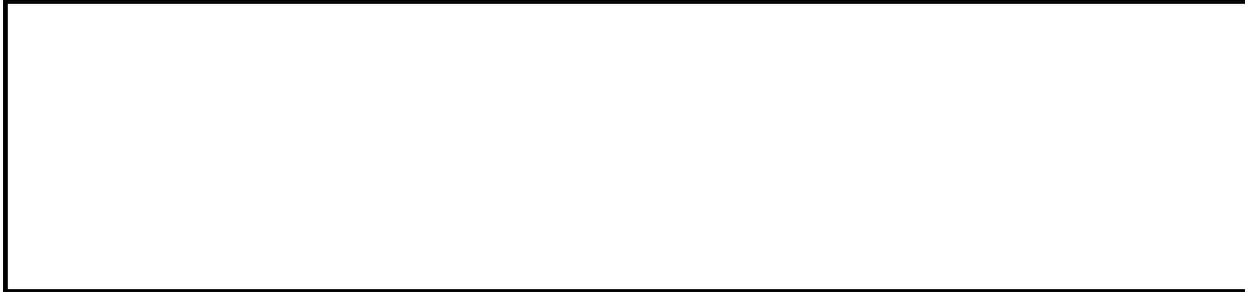
' Close the connections and clean up.
cnnConn.Close
Set cmdCommand = Nothing
Set rstRecordSet = Nothing
Set cnnConn = Nothing

```



# CreatePublisher Method

You have requested Help for a Visual Basic keyword used only on the Macintosh. For information about this keyword, consult the language reference Help included with Microsoft Office Macintosh Edition.



[Show All](#)

# CreateSummary Method

Creates a new worksheet that contains a summary report for the scenarios on the specified worksheet. **Variant**.

*expression*.CreateSummary(**ReportType**, **ResultCells**)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**ReportType** Optional [XlSummaryReportType](#).

XlSummaryReportType can be one of these XlSummaryReportType constants.

**xlSummaryPivotTable**

**xlStandardSummary** *default*

**ResultCells** Optional **Variant**. A **Range** object that represents the result cells on the specified worksheet. Normally, this range refers to one or more cells containing the formulas that depend on the changing cell values for your model — that is, the cells that show the results of a particular scenario. If this argument is omitted, there are no result cells included in the report.

## Example

This example creates a summary of the scenarios on Sheet1, with result cells in the range C4:C9 on Sheet1.

```
Worksheets("Sheet1").Scenarios.CreateSummary _  
    ResultCells := Worksheets("Sheet1").Range("C4:C9")
```



# CustomDrop Method

Sets the vertical distance (in points) from the edge of the text bounding box to the place where the callout line attaches to the text box. This distance is measured from the top of the text box unless the **AutoAttach** property is set to **True** and the text box is to the left of the origin of the callout line (the place that the callout points to), in which case the drop distance is measured from the bottom of the text box.

*expression*.**CustomDrop**(*Drop*)

*expression* Required. An expression that returns a **CalloutFormat** object.

**Drop** Required **Single**. The drop distance, in points.

## Example

This example sets the custom drop distance to 14 points, and specifies that the drop distance always be measured from the top. For the example to work, shape three must be a callout.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(3).Callout
    .CustomDrop 14
    .AutoAttach = False
End With
```



# CustomLength Method

Specifies that the first segment of the callout line (the segment attached to the text callout box) retain a fixed length whenever the callout is moved. Use the [AutomaticLength](#) method to specify that the first segment of the callout line be scaled automatically whenever the callout is moved. Applies only to callouts whose lines consist of more than one segment (types **msoCalloutThree** and **msoCalloutFour**).

*expression*.**CustomLength**(*Length*)

*expression* Required. An expression that returns a **CalloutFormat** object.

**Length** Required **Single**. The length of the first segment of the callout, in points.

## Remarks

Applying this method sets the [AutoLength](#) property to **False** and sets the [Length](#) property to the value specified for the **Length** argument.

## Example

This example toggles between an automatically scaling first segment and one with a fixed length for the callout line for shape one on myDocument. For the example to work, shape one must be a callout.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(1).Callout
    If .AutoLength Then
        .CustomLength 50
    Else
        .AutomaticLength
    End If
End With
```



# Cut Method

Cuts the object to the Clipboard or pastes it into a specified destination.

*expression.Cut(Destination)*

*expression* Required. An expression that returns an object in the Applies To list.

**Destination** Optional **Variant**. Used only with **Range** objects. The range where the object should be pasted. If this argument is omitted, the object is cut to the Clipboard.

## **Remarks**

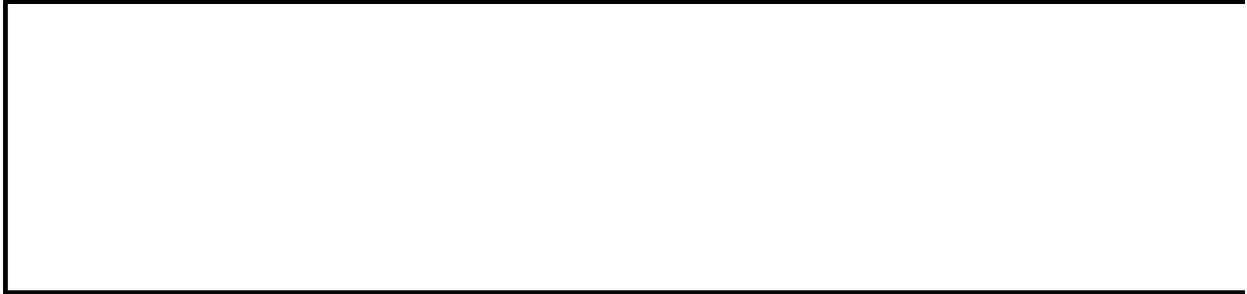
The cut range must be made up of adjacent cells.

Only embedded charts can be cut.

## Example

This example cuts the range A1:G37 on Sheet1 and places it on the Clipboard.

```
Worksheets("Sheet1").Range("A1:G37").Cut
```



# DataLabels Method

Returns an object that represents either a single data label (a [DataLabel](#) object) or a collection of all the data labels for the series (a [DataLabels](#) collection).

*expression*.**DataLabels**(*Index*)

*expression* Required. An expression that returns a **Series** object.

*Index* Optional **Variant**. The number of the data label.

## Remarks

If the series has the **Show Value** option turned on for the data labels, the returned collection can contain up to one label for each point. Data labels can be turned on or off for individual points in the series.

If the series is on an area chart and has the **Show Label** option turned on for the data labels, the returned collection contains only a single label, which is the label for the area series.

## Example

This example sets the data labels for series one in Chart1 to show their key, assuming that their values are visible when the example runs.

```
With Charts("Chart1").SeriesCollection(1)
    .HasDataLabels = True
    With .DataLabels
        .ShowLegendKey = True
        .Type = xlValue
    End With
End With
```



[Show All](#)

# DataSeries Method

Creates a data series in the specified range. **Variant**.

*expression*.**DataSeries**(*Rowcol*, *Type*, *Date*, *Step*, *Stop*, *Trend*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Rowcol** Optional **Variant**. Can be the **xlRows** or **xlColumns** constant to have the data series entered in rows or columns, respectively. If this argument is omitted, the size and shape of the range is used.

**Type** Optional [XLDataSeriesType](#).

XLDataSeriesType can be one of these XLDataSeriesType constants.

**xlAutoFill**

**xlDataSeriesLinear** *default*

**xlChronological**

**xlGrowth**

**Date** Optional [XLDataSeriesDate](#). If the **Type** argument is **xlChronological**, the **Date** argument indicates the step date unit.

XLDataSeriesDate can be one of these XLDataSeriesDate constants.

**xlDay** *default*

**xlWeekday**

**xlMonth**

**xlYear**

**Step** Optional **Variant**. The step value for the series. The default value is 1.

**Stop** Optional **Variant**. The stop value for the series. If this argument is omitted, Microsoft Excel fills to the end of the range.

**Trend** Optional **Variant**. **True** to create a linear trend or growth trend. **False** to

create a standard data series. The default value is **False**.

## Example

This example creates a series of 12 dates. The series contains the last day of every month in 1996 and is created in the range A1:A12 on Sheet1.

```
Set dateRange = Worksheets("Sheet1").Range("A1:A12")  
Worksheets("Sheet1").Range("A1").Formula = "31-JAN-1996"  
dateRange.DataSeries Type:=xlChronological, Date:=xlMonth
```



# DDEExecute Method

Runs a command or performs some other action or actions in another application by way of the specified DDE channel.

*expression*.**DDEExecute**(*Channel*, *String*)

*expression* Optional. An expression that returns an **Application** object.

**Channel** Required **Long**. The channel number returned by the [DDEInitiate](#) method.

**String** Required **String**. The message defined in the receiving application.

## Remarks

The **DDEExecute** method is designed to send commands to another application. You can also use it to send keystrokes to another application, although the [SendKeys](#) method is the preferred way to send keystrokes. The **String** argument can specify any single key combined with ALT, CTRL, or SHIFT, or any combination of those keys. Each key is represented by one or more characters, such as "a" for the character a, or "{ENTER}" for the ENTER key.

To specify characters that aren't displayed when you press the corresponding key (for example, ENTER or TAB), use the codes listed in the following table. Each code in the table represents one key on the keyboard.

Key	Code
BACKSPACE	{BACKSPACE} or {BS}
BREAK	{BREAK}
CAPS LOCK	{CAPSLOCK}
CLEAR	{CLEAR}
DELETE or DEL	{DELETE} or {DEL}
DOWN ARROW	{DOWN}
END	{END}
ENTER	~ (tilde)
ENTER (numeric keypad)	{ENTER}
ESC	{ESCAPE} or {ESC}
F1 through F15	{F1} through {F15}
HELP	{HELP}
HOME	{HOME}
INS	{INSERT}
LEFT ARROW	{LEFT}
NUM LOCK	{NUMLOCK}
PAGE DOWN	{PGDN}
PAGE UP	{PGUP}
RETURN	{RETURN}
RIGHT ARROW	{RIGHT}

SCROLL LOCK	{SCROLLLOCK}
TAB	{TAB}
UP ARROW	{UP}

You can also specify keys combined with SHIFT and/or CTRL and/or ALT. To specify a key combined with one or more of the keys just mentioned, use the following table.

**To combine a key with Precede the key code with**

SHIFT	+ (plus sign)
CTRL	^ (caret)
ALT	% (percent sign)

## Example

This example opens a channel to Word, opens the Word document Formletr.doc, and then sends the FilePrint command to WordBasic.

```
channelNumber = Application.DDEInitiate( _  
    app:="WinWord", _  
    topic:="C:\WINWORD\FORMLETR.DOC")  
Application.DDEExecute channelNumber, "[FILEPRINT]"  
Application.DDETerminate channelNumber
```



# DDEInitiate Method

Opens a DDE channel to an application.

*expression*.**DDEInitiate**(*App*, *Topic*)

*expression* Optional. An expression that returns an **Application** object.

**App** Required **String**. The application name.

**Topic** Required **String**. Describes something in the application to which you're opening a channel— usually a document of that application.

## Remarks

If successful, the **DDEInitiate** method returns the number of the open channel. All subsequent DDE functions use this number to specify the channel.

## Example

This example opens a channel to Word, opens the Word document Formletr.doc, and then sends the FilePrint command to WordBasic.

```
channelNumber = Application.DDEInitiate( _  
    app:="WinWord", _  
    topic:="C:\WINWORD\FORMLETR.DOC")  
Application.DDEExecute channelNumber, "[FILEPRINT]"  
Application.DDETerminate channelNumber
```



# DDEPoke Method

Sends data to an application.

*expression*.**DDEPoke**(*Channel*, *Item*, *Data*)

*expression* Optional. An expression that returns an **Application** object.

**Channel** Required **Long**. The channel number returned by the [DDEInitiate](#) method.

**Item** Required **Variant**. The item to which the data is to be sent.

**Data** Required **Variant**. The data to be sent to the application.

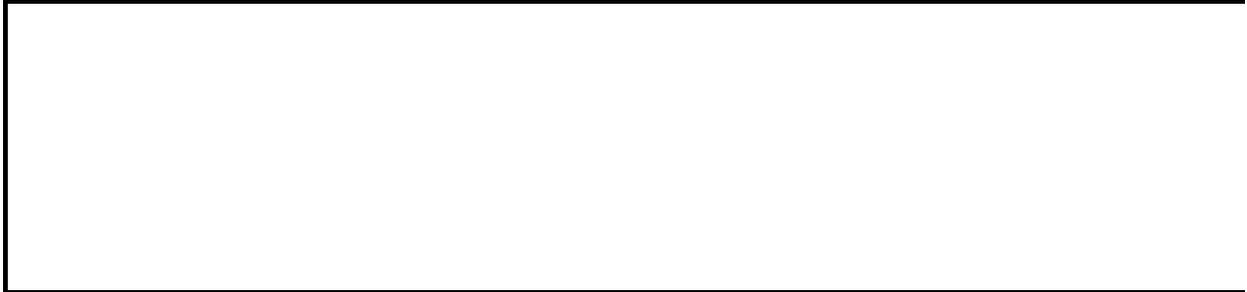
## Remarks

An error occurs if the method call doesn't succeed.

## Example

This example opens a channel to Word, opens the Word document Sales.doc, and then inserts the contents of cell A1 (on Sheet1) at the beginning of the document.

```
channelNumber = Application.DDEInitiate( _  
    app:="WinWord", _  
    topic:="C:\WINWORD\SALES.DOC")  
Set rangeToPoke = Worksheets("Sheet1").Range("A1")  
Application.DDEPoke channelNumber, "\StartOfDoc", rangeToPoke  
Application.DDETerminate channelNumber
```



# DDERequest Method

Requests information from the specified application. This method always returns an array; for more information, see the example.

*expression*.DDERequest(*Channel*, *Item*)

*expression* Optional. An expression that returns an **Application** object.

**Channel** Required **Long**. The channel number returned by the [DDEInitiate](#) method.

**Item** Required **String**. The item to be requested.

## Example

This example opens a channel to the System topic in Word and then uses the Topics item to return a list of all open documents. The list is returned in column A on Sheet1.

```
channelNumber = Application.DDEInitiate( _  
    app:="WinWord", _  
    topic:="System")  
returnList = Application.DDERequest(channelNumber, "Topics")  
For i = LBound(returnList) To UBound(returnList)  
    Worksheets("Sheet1").Cells(i, 1).Formula = returnList(i)  
Next i  
Application.DDETerminate channelNumber
```



# DDETerminate Method

Closes a channel to another application.

*expression*.**DDETerminate**(*Channel*)

*expression* Optional. An expression that returns an **Application** object.

**Channel** Required **Long**. The channel number returned by the [DDEInitiate](#) method.

## Example

This example opens a channel to Word, opens the Word document Formletr.doc, and then sends the FilePrint command to WordBasic.

```
channelNumber = Application.DDEInitiate( _  
    app:="WinWord", _  
    topic:="C:\WINWORD\FORMLETR.DOC")  
Application.DDEExecute channelNumber, "[FILEPRINT]"  
Application.DDETerminate channelNumber
```



[Show All](#)

# Delete Method

 [Delete method as it applies to the \*\*Range\*\* object.](#)

Deletes the object.

*expression*.**Delete**(*Shift*)

*expression* Required. An expression that returns a [Range](#) object.

**Shift** Optional **VARIANT**. Used only with **Range** objects. Specifies how to shift cells to replace deleted cells. Can be one of the following **XlDeleteShiftDirection** constants: **xlShiftToLeft** or **xlShiftUp**. If this argument is omitted, Microsoft Excel decides based on the shape of the range.

 [Delete method as it applies to the \*\*ListColumn\*\* object.](#)

Deletes the column of data in the list. Does not remove the column from the sheet. If the list is linked to a Microsoft Windows SharePoint Services site, the column cannot be removed from the server, and an error is generated.

*expression*.**Delete**()

*expression* Required. An expression that returns a [ListColumn](#) object.

 [Delete method as it applies to the \*\*ListObject\*\* object.](#)

Deletes the **ListObject** object and clears the cell data from the worksheet. If the list is linked to a SharePoint site, deleting it does not affect data on the server that is running Windows SharePoint Services. Any uncommitted changes made to the local list are not sent to the SharePoint list. (There is no warning that these uncommitted changes are lost.)

*expression*.**Delete**()

*expression* Required. An expression that returns a [ListObject](#) object.

 [Delete method as it applies to the \*\*ListRow\*\* object.](#)

Deletes the cells of the list row and shifts upward any remaining cells below the deleted row. You can delete rows in the list even when the list is linked to a SharePoint site. The list on the SharePoint site will not be updated, however, until you synchronize your changes.

*expression*.**Delete()**

*expression* Required. An expression that returns a [ListRow](#) object.

 [Delete method as it applies to the \*\*ShapeNodes\*\* object.](#)

Deletes the object.

*expression*.**Delete(*Index*)**

*expression* Required. An expression that returns a [ShapeNode](#) object.

*Index* Required **Integer**.

 [Delete method as it applies to the \*\*XmlMap\*\* object.](#)

Removes the specified XML map from the workbook.

*expression*.**Delete()**

*expression* Required. An expression that returns an [XmlMap](#) object.

 [Delete method as it applies to all other objects in the Applies To list.](#)

Deletes the object.

*expression*.**Delete()**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

When you delete a **Workbook** or **Worksheet**, this method displays a dialog box that prompts the user to confirm the deletion. This dialog box is displayed by default. When called on the **Workbook** or **Worksheet** objects, the **Delete** method returns a **Boolean** value that is **False** if the user clicked Cancel on the dialog box or **True** if the user clicked Delete.

Deleting a **Point** or **LegendKey** object deletes the entire series.

You can delete custom document properties, but you cannot delete a built-in document property.

You can delete cube fields only if you have created the cube fields yourself by using the **CalculatedMember.Add** method with the *xlCalculatedSet* argument.

## Example

This example deletes cells A1:D10 on Sheet1 and shifts the remaining cells to the left.

```
Worksheets("Sheet1").Range("A1:D10").Delete Shift:=xlShiftToLeft
```

This example deletes Sheet3 in the active workbook without displaying the confirmation dialog box.

```
Application.DisplayAlerts = False  
Worksheets("Sheet3").Delete  
Application.DisplayAlerts = True
```

This example sorts the data in a column of the worksheet specified and then deletes rows that contain duplicate data.

```
Sub DeleteColumnDupes(strSheetName As String, strColumnLetter As Str  
    Dim strColumnRange As String  
    Dim rngCurrentCell As Range  
    Dim rngNextCell As Range  
  
    strColumnRange = strColumnLetter & "1"  
  
    Worksheets(strSheetName).Range(strColumnRange).Sort _  
        Key1:=Worksheets(strSheetName).Range(strColumnRange)  
    Set rngCurrentCell = Worksheets(strSheetName).Range(strColumnRan  
    Do While Not IsEmpty(rngCurrentCell)  
        Set rngNextCell = rngCurrentCell.Offset(1, 0)  
        If rngNextCell.Value = rngCurrentCell.Value Then  
            rngCurrentCell.EntireRow.Delete  
        End If  
        Set rngCurrentCell = rngNextCell  
    Loop  
End Sub
```



# DeleteAll Method

Removes all users associated with access to a protected range on a worksheet.

*expression*.**DeleteAll**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel removes all users associated with access to the first protected range on the active worksheet. This example assumes the worksheet is not protected.

```
Sub UseDeleteAll()  
    Dim wksSheet As Worksheet  
    Set wksSheet = Application.ActiveSheet  
    ' Remove all users associated with access to the first protected  
    wksSheet.Protection.AllowEditRanges(1).Users.DeleteAll  
End Sub
```



# DeleteChartAutoFormat Method

Removes a custom chart autoformat from the list of available chart autoformats.

*expression*.DeleteChartAutoFormat(*Name*)

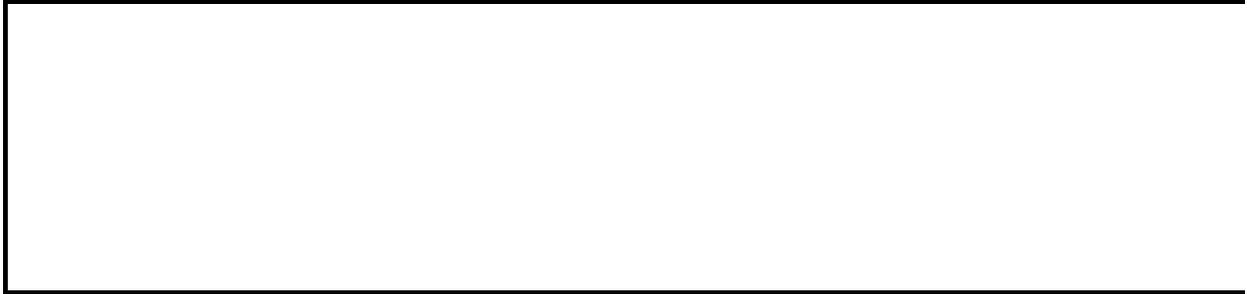
*expression* Required. An expression that returns an **Application** object.

*Name* Required **String**. The name of the custom autoformat to be removed.

## Example

This example deletes the custom autofformat named "Presentation Chart."

```
Application.DeleteChartAutoFormat name:="Presentation Chart"
```



# DeleteCustomList Method

Deletes a custom list.

*expression*.DeleteCustomList(*ListNum*)

*expression* Required. An expression that returns an **Application** object.

*ListNum* Required **Long**. The custom list number. This number must be greater than or equal to 5 (Microsoft Excel has four built-in custom lists that cannot be deleted).

## Remarks

This method generates an error if the list number is less than 5 or if there's no matching custom list.

## Example

This example deletes a custom list.

```
n = Application.GetCustomListNum(Array("cogs", "sprockets", _  
    "widgets", "gizmos"))  
Application.DeleteCustomList n
```



# DeleteNumberFormat Method

Deletes a custom number format from the workbook.

*expression*.DeleteNumberFormat(*NumberFormat*)

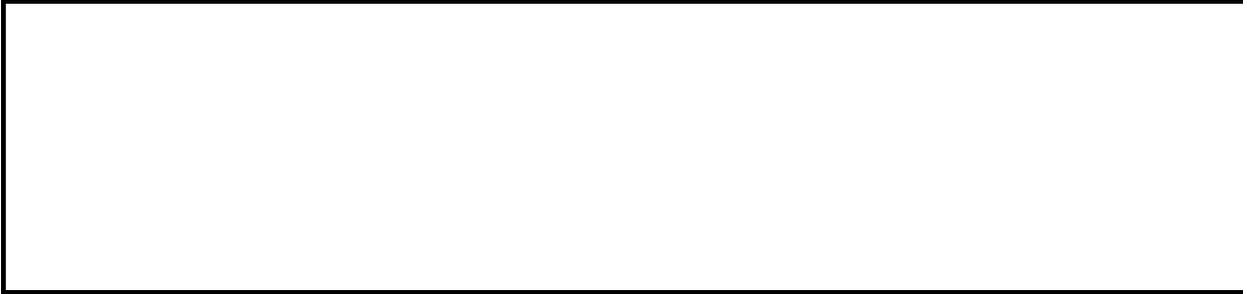
*expression* Required. An expression that returns a **Workbook** object.

*NumberFormat* Required **String**. Names the number format to be deleted.

## Example

This example deletes the number format "000-00-0000" from the active workbook.

```
ActiveWorkbook.DeleteNumberFormat("000-00-0000")
```



# DeleteReplacement Method

Deletes an entry from the array of AutoCorrect replacements.

*expression*.**DeleteReplacement**(*What*)

*expression* Required. An expression that returns an **AutoCorrect** object.

**What** Required **String**. The text to be replaced, as it appears in the row to be deleted from the array of AutoCorrect replacements. If this string doesn't exist in the array of AutoCorrect replacements, this method fails.

## Example

This example removes the word "Temperature" from the array of AutoCorrect replacements.

```
With Application.AutoCorrect  
    .DeleteReplacement "Temperature"  
End With
```



# Deselect Method

Cancels the selection for the specified chart.

*expression*.**Deselect**

*expression* Required. An expression that returns a **Chart** object.

## Example

This example is equivalent to pressing ESC while working on the active chart. The example should be run on a chart that has a component (such as an axis) selected.

ActiveChart.**Deselect**



# DialogBox Method

Displays a dialog box defined by a dialog box definition table on a Microsoft Excel 4.0 macro sheet. Returns the number of the chosen control, or returns **False** if the user clicks the **Cancel** button.

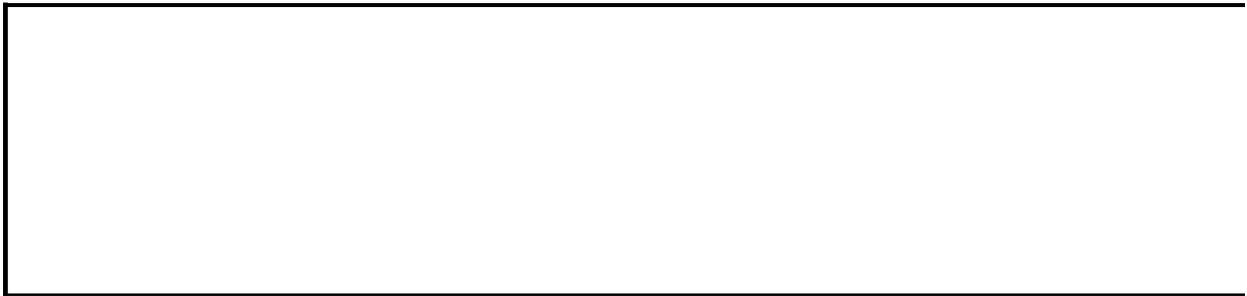
*expression*.**DialogBox**

*expression* Required. An expression that returns a **Range** object. The **Range** must refer to a dialog box definition table on a Microsoft Excel 4.0 macro sheet.

## Example

This example runs a Microsoft Excel 4.0 dialog box and then displays the return value in a message box. The dialogRange variable refers to the dialog box definition table on the Microsoft Excel 4.0 macro sheet named "Macro1."

```
Set dialogRange = Excel4MacroSheets("Macro1").Range("myDialogBox")  
result = dialogRange.DialogBox  
MsgBox result
```



# Dirty Method

Designates a range to be recalculated when the next recalculation occurs.

*expression*.**Dirty**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

The [Calculate](#) method forces the specified range to be recalculated, for cells that Microsoft Excel understands as needing recalculation.

If the application is in manual calculation mode, using the **Dirty** method instructs Excel to identify the specified cell to be recalculated. If the application is in automatic calculation mode, using the **Dirty** method instructs Excel to perform a recalculation.

## Example

In this example, Microsoft Excel enters a formula in cell A3, saves the changes, and then recalculates cell A3.

```
Sub UseDirtyMethod()  
  
    MsgBox "Two values and a formula will be entered."  
    Range("A1").Value = 1  
    Range("A2").Value = 2  
    Range("A3").Formula = "=A1+A2"  
  
    ' Save the changes made to the worksheet.  
    Application.DisplayAlerts = False  
    Application.Save  
    MsgBox "Changes saved."  
  
    ' Force a recalculation of range A3.  
    Application.Range("A3").Dirty  
    MsgBox "Try to close the file without saving and a dialog box wi  
  
End Sub
```

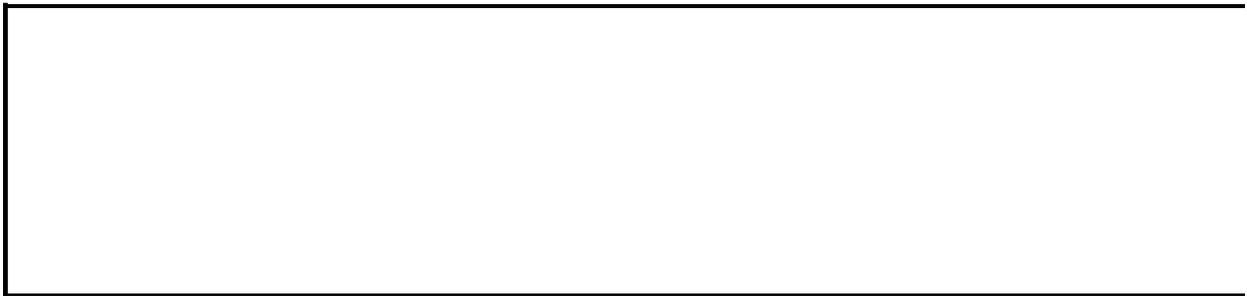


# Disconnect Method

Instructs the real-time data server (RTD) to disconnect from the specified **IRTDUpdateEvent** object.

*expression*.**Disconnect()**

*expression* Required. An expression that returns an **IRTDUpdateEvent** object.



# DisconnectData Method

Notifies a real-time data (RTD) server application that a topic is no longer in use.

*expression*.**DisconnectData**(*TopicID*)

*expression* Required. An expression that returns an **IRtdServer** object.

**TopicID** Required **Long**. A unique value assigned to the topic assigned by Microsoft Excel.



# DisplayXMLSourcePane Method

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Opens the **XML Source** task pane and displays the XML map specified by the ***XmlMap*** argument.

*expression*.**DisplayXMLSourcePane**(*XmlMap*)

*expression* Required. An expression that returns an [Application](#) object.

***XmlMap*** Optional [XmlMap](#) object. The XML map to display in the task pane.

## Remarks

You can use the following code to hide the **XML Source** task pane.

```
Application.CommandBars("Task Pane").Visible = False
```

## Example

The following example adds an XML map named Customers to the active workbook, and then displays the XML map in the **XML Source** task pane.

```
Sub DisplayXMLMap()  
    Dim objCustomer As XmlMap  
  
    Set objCustomer = ActiveWorkbook.XmlMaps.Add( _  
        "Customers.xsd", "Root")  
  
        objCustomer.Name = "Customers"  
  
    Application.DisplayXMLSourcePane  
        objCustomer  
End Sub
```



[Show All](#)

# Distribute Method

Horizontally or vertically distributes the shapes in the specified range of shapes.

*expression*.**Distribute**(*DistributeCmd*, *RelativeTo*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*DistributeCmd* Required [MsoDistributeCmd](#). Specifies whether shapes in the range are to be distributed horizontally or vertically.

MsoDistributeCmd can be one of these MsoDistributeCmd constants.

**msoDistributeHorizontally**

**msoDistributeVertically**

*RelativeTo* Required [MsoTriState](#). Not used in Microsoft Excel. Must be **False**.

MsoTriState can be one of these MsoTriState constants.

**msoCTrue**

**msoFalse**

**msoTriStateMixed**

**msoTriStateToggle**

**msoTrue**

## Example

This example defines a shape range that contains all the AutoShapes on myDocument and then horizontally distributes the shapes in this range. The leftmost shape retains its position.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes
    numShapes = .Count
    If numShapes > 1 Then
        numAutoShapes = 0
        ReDim autoShpArray(1 To numShapes)
        For i = 1 To numShapes
            If .Item(i).Type = msoAutoShape Then
                numAutoShapes = numAutoShapes + 1
                autoShpArray(numAutoShapes) = .Item(i).Name
            End If
        Next
        If numAutoShapes > 1 Then
            ReDim Preserve autoShpArray(1 To numAutoShapes)
            Set asRange = .Range(autoShpArray)
            asRange.Distribute msoDistributeHorizontally, False
        End If
    End If
End With
```



# DoubleClick Method

Equivalent to double-clicking the active cell.

*expression*.**DoubleClick**

*expression* Required. An expression that returns an **Application** object.

## Example

This example double-clicks the active cell on Sheet1.

```
Worksheets("Sheet1").Activate  
Application.DoubleClick
```



# DoughnutGroups Method

On a 2-D chart, returns an object that represents either a single doughnut chart group (a [ChartGroup](#) object) or a collection of the doughnut chart groups (a [ChartGroups](#) collection).

*expression*.**DoughnutGroups**(*Index*)

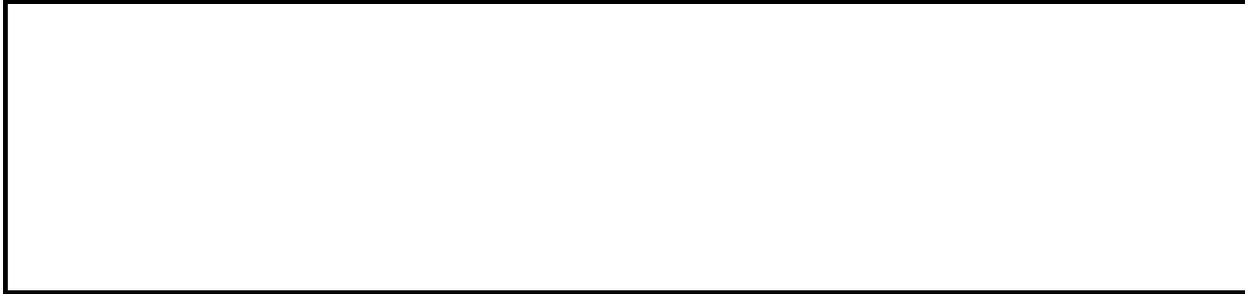
*expression* Required. An expression that returns a **Chart** object.

*Index* Optional **Variant**. Specifies the chart group.

## Example

This example sets the starting angle for doughnut group one in Chart1.

```
Charts("Chart1").DoughnutGroups(1).FirstSliceAngle = 45
```



[Show All](#)

# DragOff Method

Drags a page break out of the print area.

*expression*.**DragOff**(*Direction*, *RegionIndex*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Direction** Required [XlDirection](#). The direction in which the page break is dragged.

XlDirection can be one of these XlDirection constants.

**xlDown**

**xlToRight**

**xlToLeft**

**xlUp**

**RegionIndex** Required **Long**. The print-area region index for the page break (the region where the mouse pointer is located when the mouse button is pressed if the user drags the page break). If the print area is contiguous, there's only one print region. If the print area is discontinuous, there's more than one print region.

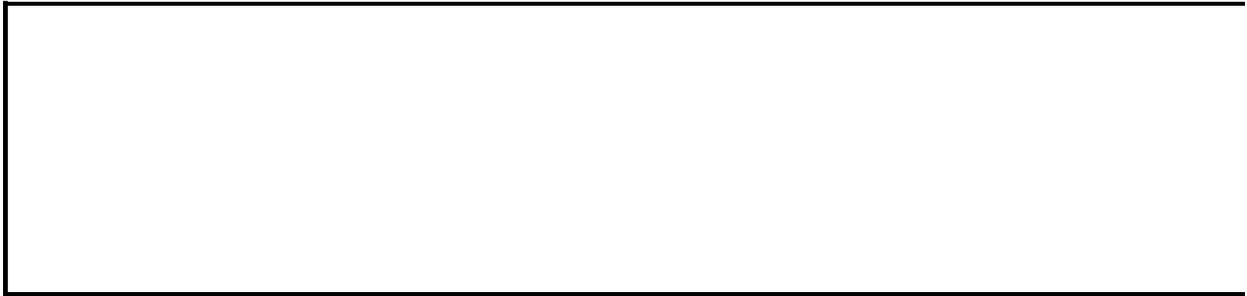
## Remarks

This method exists primarily for the macro recorder. You can use the **Delete** method to delete a page break in Visual Basic.

## Example

This example deletes vertical page break one from the active sheet by dragging it off the right edge of print region one.

```
ActiveSheet.VPageBreaks(1).DragOff xlToRight, 1
```



# Duplicate Method

Duplicates the object and returns a reference to the new copy.

*expression*.**Duplicate**

*expression* Required. An expression that returns an object in the Applies To list.

## Example

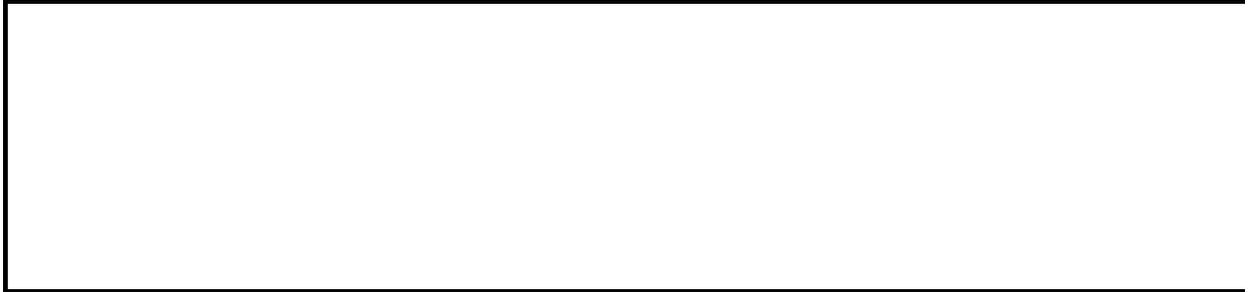
This example duplicates embedded chart one on Sheet1 and then selects the copy.

```
Set dChart = Worksheets("Sheet1").ChartObjects(1).Duplicate  
dChart.Select
```



# **EditionOptions Method**

You have requested Help for a Visual Basic keyword used only on the Macintosh. For information about this keyword, consult the language reference Help included with Microsoft Office Macintosh Edition.



# EndConnect Method

Attaches the end of the specified connector to a specified shape. If there's already a connection between the end of the connector and another shape, that connection is broken. If the end of the connector isn't already positioned at the specified connecting site, this method moves the end of the connector to the connecting site and adjusts the size and position of the connector. Use the **BeginConnect** method to attach the beginning of the connector to a shape.

*expression*.**EndConnect**(*ConnectedShape*, *ConnectionSite*)

*expression* Required. An expression that returns a **ConnectorFormat** object.

**ConnectedShape** Required **Shape** object. The shape to attach the end of the connector to. The specified **Shape** object must be in the same **Shapes** collection as the connector.

**ConnectionSite** Required **Long**. A connection site on the shape specified by **ConnectedShape**. Must be an integer between 1 and the integer returned by the **ConnectionSiteCount** property of the specified shape. If you want the connector to automatically find the shortest path between the two shapes it connects, specify any valid integer for this argument and then use the [RerouteConnections](#) method after the connector is attached to shapes at both ends.

## Remarks

When you attach a connector to an object, the size and position of the connector are automatically adjusted, if necessary.

## Example

This example adds two rectangles to myDocument and connects them with a curved connector. Notice that the **RerouteConnections** method makes it irrelevant what values you supply for the **ConnectionSite** arguments used with the **BeginConnect** and **EndConnect** methods.

```
Set myDocument = Worksheets(1)
Set s = myDocument.Shapes
Set firstRect = s.AddShape(msoShapeRectangle, 100, 50, 200, 100)
Set secondRect = s.AddShape(msoShapeRectangle, 300, 300, 200, 100)
Set c = s.AddConnector(msoConnectorCurve, 0, 0, 100, 100)
With c.ConnectorFormat
    .BeginConnect ConnectedShape:=firstRect, ConnectionSite:=1
    .EndConnect ConnectedShape:=secondRect, ConnectionSite:=1
    c.RerouteConnections
End With
```



# EndDisconnect Method

Detaches the end of the specified connector from the shape it's attached to. This method doesn't alter the size or position of the connector: the end of the connector remains positioned at a connection site but is no longer connected. Use the [BeginDisconnect](#) method to detach the beginning of the connector from a shape.

*expression*.**EndDisconnect**

*expression* Required. An expression that returns a **ConnectorFormat** object.

## Example

This example adds two rectangles to myDocument, attaches them with a connector, automatically reroutes the connector along the shortest path, and then detaches the connector from the rectangles.

```
Set myDocument = Worksheets(1)
Set s = myDocument.Shapes
Set firstRect = s.AddShape(msoShapeRectangle, 100, 50, 200, 100)
Set secondRect = s.AddShape(msoShapeRectangle, 300, 300, 200, 100)
set c = s.AddConnector(msoConnectorCurve, 0, 0, 0, 0)
with c.ConnectorFormat
    .BeginConnect firstRect, 1
    .EndConnect secondRect, 1
    c.RerouteConnections
    .BeginDisconnect
    .EndDisconnect
End With
```



# EndReview Method

Terminates a review of a file that has been sent for review using the [SendForReview](#) method.

*expression*.**EndReview**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example terminates the review of the active workbook. When executed, this procedure displays a message asking if you want to end the review. This example assumes the active workbook has been sent for review.

```
Sub EndWorkbookRev()  
    ActiveWorkbook.EndReview  
End Sub
```



[Show All](#)

# ErrorBar Method

Applies error bars to the series. **Variant**.

*expression*.**ErrorBar**(*Direction*, *Include*, *Type*, *Amount*, *MinusValues*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Direction** Required [XLErrorBarDirection](#). The error bar direction.

XLErrorBarDirection can be one of these XLErrorBarDirection constants.

**xlX**

**xlY** *default*

**Include** Required [XLErrorBarInclude](#). The error bar parts to include.

XLErrorBarInclude can be one of these XLErrorBarInclude constants.

**xLErrorBarIncludeBoth** *default*

**xLErrorBarIncludeNone**

**xLErrorBarIncludeMinusValues**

**xLErrorBarIncludePlusValues**

**Type** Required [XLErrorBarType](#). The error bar type.

XLErrorBarType can be one of these XLErrorBarType constants.

**xLErrorBarTypeCustom**

**xLErrorBarTypePercent**

**xLErrorBarTypeStError**

**xLErrorBarTypeFixedValue**

**xLErrorBarTypeStDev**

**Amount** Optional **Variant**. The error amount. Used for only the positive error amount when *Type* is **xLErrorBarTypeCustom**.

***MinusValues*** Optional **Variant**. The negative error amount when ***Type*** is **xlErrorBarTypeCustom**.

## Example

This example applies standard error bars in the Y direction for series one in Chart1. The error bars are applied in the positive and negative directions. The example should be run on a 2-D line chart.

```
Charts("Chart1").SeriesCollection(1).ErrorBar _  
    Direction:=xlY, Include:=xlErrorBarIncludeBoth, _  
    Type:=xlErrorBarTypeStError
```



# Evaluate Method

Converts a Microsoft Excel name to an object or a value.

*expression*.**Evaluate**(*Name*)

*expression* Optional for **Application**, required for **Chart**, **DialogSheet**, and **Worksheet**. An expression that returns an object in the Applies To list.

*Name* Required **String**. The name of the object, using the naming convention of Microsoft Excel.

## Remarks

The following types of names in Microsoft Excel can be used with this method:

- A1-style references. You can use any reference to a single cell in A1-style notation. All references are considered to be absolute references.
- Ranges. You can use the range, intersect, and union operators (colon, space, and comma, respectively) with references.
- Defined names. You can specify any name in the language of the macro.
- External references. You can use the ! operator to refer to a cell or to a name defined in another workbook— for example, Evaluate(" [BOOK1.XLS]Sheet1!A1").

**Note** Using square brackets (for example, "[A1:C5]") is identical to calling the **Evaluate** method with a string argument. For example, the following expression pairs are equivalent.

```
[a1].Value = 25  
Evaluate("A1").Value = 25
```

```
trigVariable = [SIN(45)]  
trigVariable = Evaluate("SIN(45)")
```

```
Set firstCellInSheet = Workbooks("BOOK1.XLS").Sheets(4).[A1]  
Set firstCellInSheet = _  
    Workbooks("BOOK1.XLS").Sheets(4).Evaluate("A1")
```

The advantage of using square brackets is that the code is shorter. The advantage of using **Evaluate** is that the argument is a string, so you can either construct the string in your code or use a Visual Basic variable.

## Example

This example turns on bold formatting in cell A1 on Sheet1.

```
Worksheets("Sheet1").Activate  
boldCell = "A1"  
Application.Evaluate(boldCell).Font.Bold = True
```



# ExclusiveAccess Method

Assigns the current user exclusive access to the workbook that's open as a shared list.

*expression*.**ExclusiveAccess**

*expression* Required. An expression that returns a **Workbook** object.

## Remarks

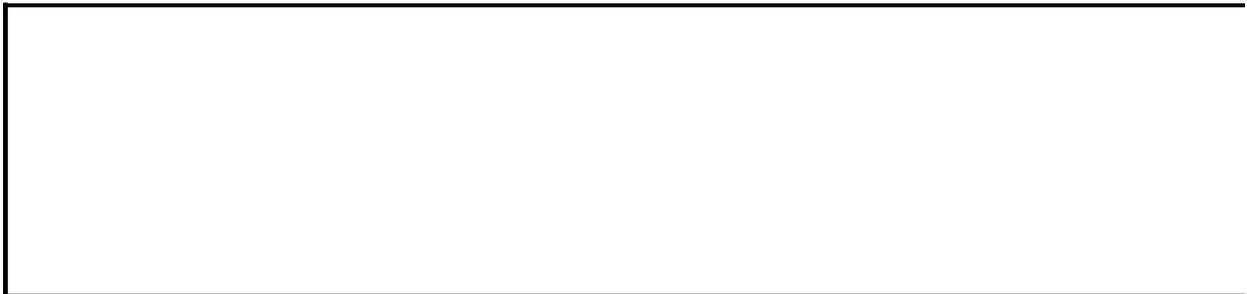
The **ExclusiveAccess** method saves any changes you've made to the workbook and requires other users who have the workbook open to save their changes to a different file.

If the specified workbook isn't open as a shared list, this method fails. To determine whether a workbook is open as a shared list, use the **MultiUserEditing** property.

## Example

This example determines whether the active workbook is open as a shared list. If it is, the example gives the current user exclusive access.

```
If ActiveWorkbook.MultiUserEditing Then  
    ActiveWorkbook.ExclusiveAccess  
End If
```



# Execute Method

Activates a smart tag action that is associated with the smart tag type on a cell.

*expression*.**Execute**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example inserts a refreshable stock quote for the ticker symbol "MSFT" and it assumes the host system is connected to the Internet.

```
Sub ExecuteASmartTag()  
    Dim strAction As String  
  
    strAction = "Insert refreshable stock price..."  
  
    ' Enable smart tags to be embedded and recognized.  
    ActiveWorkbook.SmartTagOptions.EmbedSmartTags = True  
    Application.SmartTagRecognizers.Recognize = True  
  
    ' Invoke a smart tag for the Microsoft ticker symbol.  
    With Range("A1")  
        .Formula = "MSFT"  
        .SmartTags.Add( _  
            "urn:schemas-microsoft-com:office:smartsheet#stockticker"  
            .SmartTagActions(strAction).Execute  
        End With  
    End Sub
```



# ExecuteExcel4Macro Method

Runs a Microsoft Excel 4.0 macro function and then returns the result of the function. The return type depends on the function.

*expression*.ExecuteExcel4Macro(**String**)

*expression* Optional. An expression that returns an **Application** object.

**String** Required **String**. A Microsoft Excel 4.0 macro language function without the equal sign. All references must be given as R1C1 strings. If **String** contains embedded double quotation marks, you must double them. For example, to run the macro function =MID("sometext",1,4), **String** would have to be "MID(""sometext"",1,4)".

## Remarks

The Microsoft Excel 4.0 macro isn't evaluated in the context of the current workbook or sheet. This means that any references should be external and should specify an explicit workbook name. For example, to run the Microsoft Excel 4.0 macro "My\_Macro" in Book1 you must use "Book1!My\_Macro()". If you don't specify the workbook name, this method fails.

## Example

This example runs the **GET.CELL(42)** macro function on cell C3 on Sheet1 and then displays the result in a message box. The **GET.CELL(42)** macro function returns the horizontal distance from the left edge of the active window to the left edge of the active cell. This macro function has no direct Visual Basic equivalent.

```
Worksheets("Sheet1").Activate  
Range("C3").Select  
MsgBox ExecuteExcel4Macro("GET.CELL(42)")
```



[Show All](#)

# Export Method

 [Export method as it applies to the \*\*Chart\*\* object.](#)

Exports the chart in a graphic format. Returns **Boolean**.

*expression*.**Export**(*Filename*, *FilterName*, *Interactive*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Filename** Required **String**. The name of the exported file.

**FilterName** Optional **Variant**. The language-independent name of the graphic filter as it appears in the registry.

**Interactive** Optional **Variant**. Optional **Variant**. **True** to display the dialog box that contains the filter-specific options. If this argument is **False**, Microsoft Excel uses the default values for the filter. The default value is **False**.

 [Export method as it applies to the \*\*XmlMap\*\* object.](#)

Exports the contents of cells mapped to the specified **XmlMap** object to an XML data file. Returns [XlXmlExportResult](#).

XlXmlExportResult can be one of the following XlXmlExportResult constants

**xlXmlExportSuccess** The XML data file was successfully exported.

**xlXmlExportValidationFailed** The contents of the XML data file do not match the specified schema map.

.

*expression*.**Export**(*Url*, *Overwrite* )

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Url** Required **String**. The path and filename of the XML data file to export to.

**Overwrite** Required **Boolean**. Set to **True** to overwrite the file specified in the **URL** parameter if the file exists. The default value is **False**.

## Remarks

Use the **ExportXML** method to export the contents of the mapped cells to a **String** variable.

## Example

This example exports chart one as a GIF file.

```
Worksheets("Sheet1").ChartObjects(1) _  
.Chart.Export _  
    FileName:="current_sales.gif", FilterName:="GIF"
```



[Show All](#)

# ExportXml Method

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Exports the contents of cells mapped to the specified [XmlMap](#) object to a **String** variable. Returns [XlXmlExportResult](#).

**XlXmlExportResult** can be one of the following **XlXmlExportResult** constants:

**xlXmlExportSuccess** The XML data file was successfully exported.

**xlXmlExportValidationFailed** The contents of the XML data file do not match the specified schema map.

*expression*.**ExportXml(Data)**

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Data** Required **String**. The variable to export the data to.

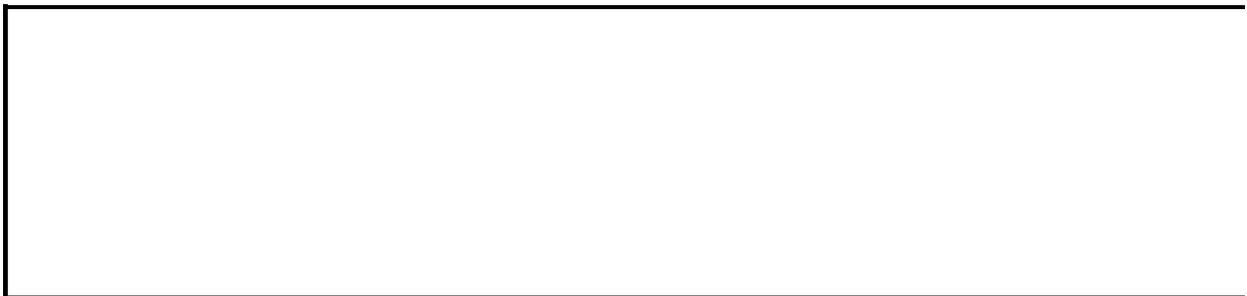
## Remarks

To export the contents of the mapped cells to an XML data file, use the [Export](#) method.

## Example

The following example exports the contents of the cells mapped to the "Contacts" schema map to a variable named strContactData.

```
Sub ExportToString()  
    Dim strContactData As String  
  
    ActiveWorkbook.XmlMaps("Contacts").ExportXml Data:=strContactDat  
End Sub
```



# Extend Method

Adds new data points to an existing series collection. **Variant**

*expression*.**Extend**(*Source*, *Rowcol*, *CategoryLabels*)

*expression* Required. An expression that returns a **SeriesCollection** object.

**Source** Required **Variant**. The new data to be added to the **SeriesCollection** object, either as a **Range** object or an array of data points.

**Rowcol** Optional **Variant**. Ignored if **Source** is an array. Specifies whether the new values are in the rows or columns of the given range source. Can be one of the following **XIRowCol** constants: **xlRows** or **xlColumns**. If this argument is omitted, Microsoft Excel attempts to determine where the values are by the size and orientation of the selected range or by the dimensions of the array.

**CategoryLabels** Optional **Variant**. Ignored if **Source** is an array. **True** to have the first row or column contain the name of the category labels. **False** to have the first row or column contain the first data point of the series. If this argument is omitted, Microsoft Excel attempts to determine the location of the category label from the contents of the first row or column.

## Remarks

This method is not available for PivotChart reports.

## Example

This example extends the series on Chart1 by adding the data in cells B1:B6 on Sheet1.

```
Charts("Chart1").SeriesCollection.Extend _  
    Source:=Worksheets("Sheet1").Range("B1:B6")
```



[Show All](#)

# FillAcrossSheets Method

Copies a range to the same area on all other worksheets in a collection.

*expression*.FillAcrossSheets(**Range**, **Type**)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Range** Required **Range** object. The range to fill on all the worksheets in the collection. The range must be from a worksheet within the collection.

**Type** Optional [XIFillWith](#). Specifies how to copy the range.

XIFillWith can be one of these XIFillWith constants.

**xIFillWithAll** *default*

**xIFillWithContents**

**xIFillWithFormats**

## Example

This example fills the range A1:C5 on Sheet1, Sheet5, and Sheet7 with the contents of the same range on Sheet1.

```
x = Array("Sheet1", "Sheet5", "Sheet7")  
Sheets(x).FillAcrossSheets _  
    Worksheets("Sheet1").Range("A1:C5")
```



# FillDown Method

Fills down from the top cell or cells in the specified range to the bottom of the range. The contents and formatting of the cell or cells in the top row of a range are copied into the rest of the rows in the range.

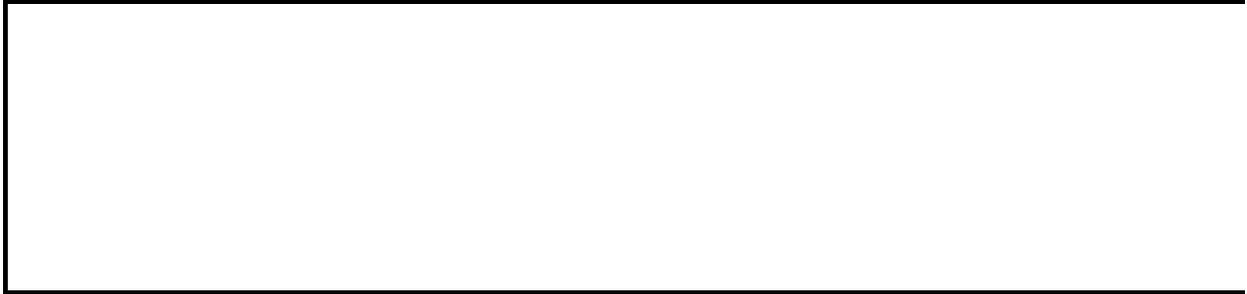
*expression*.**FillDown**

*expression* Required. An expression that returns a **Range** object.

## Example

This example fills the range A1:A10 on Sheet1, based on the contents of cell A1.

```
Worksheets("Sheet1").Range("A1:A10").FillDown
```



# FillLeft Method

Fills left from the rightmost cell or cells in the specified range. The contents and formatting of the cell or cells in the rightmost column of a range are copied into the rest of the columns in the range.

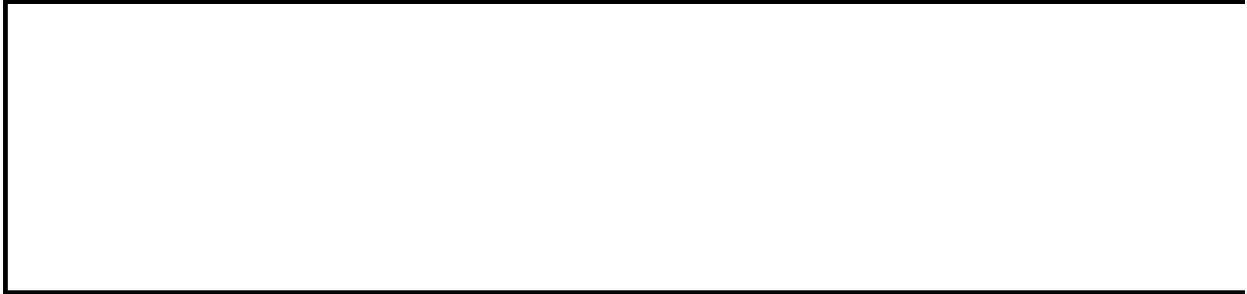
*expression*.**FillLeft**

*expression* Required. An expression that returns a **Range** object.

## Example

This example fills the range A1:M1 on Sheet1, based on the contents of cell M1.

```
Worksheets("Sheet1").Range("A1:M1").FillLeft
```



# FillRight Method

Fills right from the leftmost cell or cells in the specified range. The contents and formatting of the cell or cells in the leftmost column of a range are copied into the rest of the columns in the range.

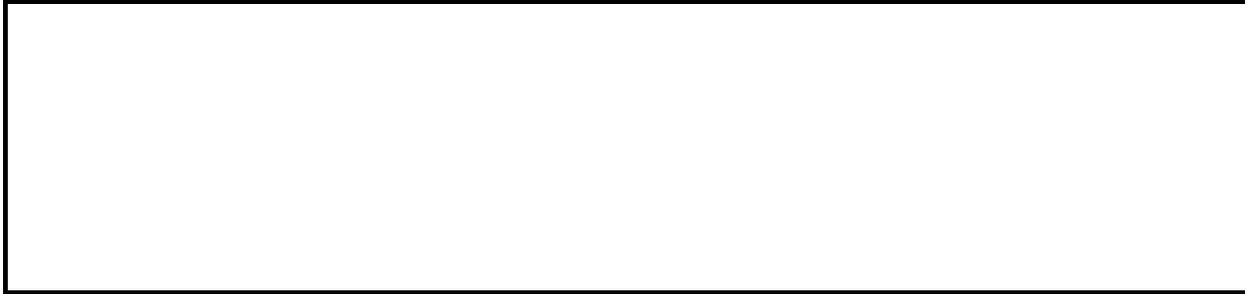
*expression*.**FillRight**

*expression* Required. An expression that returns a **Range** object.

## Example

This example fills the range A1:M1 on Sheet1, based on the contents of cell A1.

```
Worksheets("Sheet1").Range("A1:M1").FillRight
```



# FillUp Method

Fills up from the bottom cell or cells in the specified range to the top of the range. The contents and formatting of the cell or cells in the bottom row of a range are copied into the rest of the rows in the range.

*expression*.**FillUp**

*expression* Required. An expression that returns a **Range** object.

## Example

This example fills the range A1:A10 on Sheet1, based on the contents of cell A10.

```
Worksheets("Sheet1").Range("A1:A10").FillUp
```



[Show All](#)

# Find Method

 [Find method as it applies to the \*\*WorksheetFunction\*\* object.](#)

Finds specific information in a worksheet.

*expression*.**Find**(*Arg1*, *Arg2*, *Arg3*)

*expression* Required. An expression that returns a **WorksheetFunction** object.

**Arg1** Required **String**. The name of the worksheet.

**Arg2** Required **String**. The name of the range.

**Arg3** Optional **VARIANT**. The name of an argument to refine the search.

 [Find method as it applies to the \*\*Range\*\* object.](#)

Finds specific information in a range, and returns a **Range** object that represents the first cell where that information is found. Returns **Nothing** if no match is found. Doesn't affect the selection or the active cell.

For information about using the **Find** worksheet function in Visual Basic, see [Using Worksheet Functions in Visual Basic](#).

*expression*.**Find**(*What*, *After*, *LookIn*, *LookAt*, *SearchOrder*, *SearchDirection*, *MatchCase*, *MatchByte*, *SearchFormat*)

*expression* Required. An expression that returns a **Range** object.

**What** Required **VARIANT**. The data to search for. Can be a string or any Microsoft Excel data type.

**After** Optional **VARIANT**. The cell after which you want the search to begin. This corresponds to the position of the active cell when a search is done from the user interface. Note that **After** must be a single cell in the range. Remember that the search begins *after* this cell; the specified cell isn't searched until the method

wraps back around to this cell. If you don't specify this argument, the search starts after the cell in the upper-left corner of the range.

**LookIn** Optional **Variant**. The type of information.

**LookAt** Optional **Variant**. Can be one of the following **XILookAt** constants: **xlWhole** or **xlPart**.

**SearchOrder** Optional **Variant**. Can be one of the following **XISearchOrder** constants: **xlByRows** or **xlByColumns**.

**SearchDirection** Optional [XISearchDirection](#). The search direction.

XISearchDirection can be one of these XISearchDirection constants.

**xlNext** *default*

**xlPrevious**

**MatchCase** Optional **Variant**. **True** to make the search case sensitive. The default value is **False**.

**MatchByte** Optional **Variant**. Used only if you've selected or installed double-byte language support. **True** to have double-byte characters match only double-byte characters. **False** to have double-byte characters match their single-byte equivalents.

**SearchFormat** Optional **Variant**. The search format.

## Remarks

The settings for *LookIn*, *LookAt*, *SearchOrder*, and *MatchByte* are saved each time you use this method. If you don't specify values for these arguments the next time you call the method, the saved values are used. Setting these arguments changes the settings in the **Find** dialog box, and changing the settings in the **Find** dialog box changes the saved values that are used if you omit the arguments. To avoid problems, set these arguments explicitly each time you use this method.

You can use the [FindNext](#) and [FindPrevious](#) methods to repeat the search.

When the search reaches the end of the specified search range, it wraps around to the beginning of the range. To stop a search when this wraparound occurs, save the address of the first found cell, and then test each successive found-cell address against this saved address.

To find cells that match more complicated patterns, use a **For Each...Next** statement with the **Like** operator. For example, the following code searches for all cells in the range A1:C5 that use a font whose name starts with the letters Cour. When Microsoft Excel finds a match, it changes the font to Times New Roman.

```
For Each c In [A1:C5]
    If c.Font.Name Like "Cour*" Then
        c.Font.Name = "Times New Roman"
    End If
Next
```

## Example

This example finds all cells in the range A1:A500 on worksheet one that contain the value 2 and changes it to 5.

```
With Worksheets(1).Range("a1:a500")
    Set c = .Find(2, lookin:=xlValues)
    If Not c Is Nothing Then
        firstAddress = c.Address
        Do
            c.Value = 5
            Set c = .FindNext(c)
        Loop While Not c Is Nothing And c.Address <> firstAddress
    End If
End With
```



# FindFile Method

Displays the **Open** dialog box.

*expression*.**FindFile**

*expression* Required. An expression that returns an **Application** object.

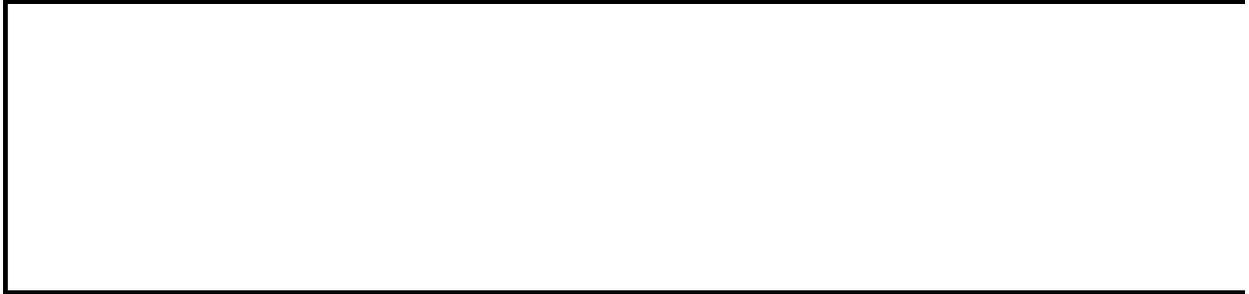
## Remarks

This method displays the **Open** dialog box and allows the user to open a file. If a new file is opened successfully, this method returns **True**. If the user cancels the dialog box, this method returns **False**.

## Example

This example displays the **Open** dialog box.

Application.**FindFile**



# FindNext Method

Continues a search that was begun with the [Find](#) method. Finds the next cell that matches those same conditions and returns a [Range](#) object that represents that cell. Doesn't affect the selection or the active cell.

*expression*.**FindNext**(*After*)

*expression* Required. An expression that returns a **Range** object.

**After** Optional **Variant**. The cell after which you want to search. This corresponds to the position of the active cell when a search is done from the user interface. Note that **After** must be a single cell in the range. Remember that the search begins *after* this cell; the specified cell isn't searched until the method wraps back around to this cell. If this argument isn't specified, the search starts after the cell in the upper-left corner of the range.

## Remarks

When the search reaches the end of the specified search range, it wraps around to the beginning of the range. To stop a search when this wraparound occurs, save the address of the first found cell, and then test each successive found-cell address against this saved address.

## Example

This example finds all cells in the range A1:A500 that contain the value 2 and changes their values to 5.

```
With Worksheets(1).Range("a1:a500")
  Set c = .Find(2, lookin:=xlValues)
  If Not c Is Nothing Then
    firstAddress = c.Address
    Do
      c.Value = 5
      Set c = .FindNext(c)
    Loop While Not c Is Nothing And c.Address <> firstAddress
  End If
End With
```



# FindPrevious Method

Continues a search that was begun with the [Find](#) method. Finds the previous cell that matches those same conditions and returns a [Range](#) object that represents that cell. Doesn't affect the selection or the active cell.

*expression*.**FindPrevious**(*After*)

*expression* Required. An expression that returns a **Range** object.

**After** Optional **Variant**. The cell before which you want to search. This corresponds to the position of the active cell when a search is done from the user interface. Note that **After** must be a single cell in the range. Remember that the search begins *before* this cell; the specified cell isn't searched until the method wraps back around to this cell. If this argument isn't specified, the search starts before the upper- left cell in the range.

## Remarks

When the search reaches the beginning of the specified search range, it wraps around to the end of the range. To stop a search when this wraparound occurs, save the address of the first found cell, and then test each successive found-cell address against this saved address.

## Example

This example shows how the **FindPrevious** method is used with the **Find** and **FindNext** methods. Before running this example, make sure that Sheet1 contains at least two occurrences of the word "Phoenix" in column B.

```
Set fc = Worksheets("Sheet1").Columns("B").Find(what:="Phoenix")
MsgBox "The first occurrence is in cell " & fc.Address
Set fc = Worksheets("Sheet1").Columns("B").FindNext(after:=fc)
MsgBox "The next occurrence is in cell " & fc.Address
Set fc = Worksheets("Sheet1").Columns("B").FindPrevious(after:=fc)
MsgBox "The previous occurrence is in cell " & fc.Address
```



[Show All](#)

# Flip Method

Flips the specified shape around its horizontal or vertical axis.

*expression*.**Flip**(*FlipCmd*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*FlipCmd* Required [MsoFlipCmd](#). Specifies whether the shape is to be flipped horizontally or vertically

MsoFlipCmd can be one of these MsoFlipCmd constants.

**msoFlipHorizontal**

**msoFlipVertical**

## Example

This example adds a triangle to myDocument, duplicates the triangle, and then flips the duplicate triangle vertically and makes it red.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddShape(msoShapeRightTriangle, _
    10, 10, 50, 50).Duplicate
    .Fill.ForeColor.RGB = RGB(255, 0, 0)
    .Flip msoFlipVertical
End With
```



# Follow Method

Displays a cached document, if it's already been downloaded. Otherwise, this method resolves the hyperlink, downloads the target document, and displays the document in the appropriate application.

*expression*.**Follow**(*NewWindow*, *AddHistory*, *ExtraInfo*, *Method*, *HeaderInfo*)

*expression* Required. An expression that returns a **Hyperlink** object.

**NewWindow** Optional **Variant**. **True** to display the target application in a new window. The default value is **False**.

**AddHistory** Optional **Variant**. Not used. Reserved for future use.

**ExtraInfo** Optional **Variant**. A **String** or byte array that specifies additional information for HTTP to use to resolve the hyperlink. For example, you can use **ExtraInfo** to specify the coordinates of an image map, the contents of a form, or a FAT file name.

**Method** Optional **Variant**. Specifies the way **ExtraInfo** is attached. Can be one of the following **MsoExtraInfoMethod** constants.

## Constant

## Description

**msoMethodGet** **ExtraInfo** is a **String** that's appended to the address.

**msoMethodPost** **ExtraInfo** is posted as a **String** or byte array.

**HeaderInfo** Optional **Variant**. A **String** that specifies header information for the HTTP request. The default value is an empty string.

## Example

This example loads the document attached to the hyperlink on shape one on worksheet one.

```
Worksheets(1).Shapes(1).Hyperlink.Follow NewWindow:=True
```



[Show All](#)

# FollowHyperlink Method

Displays a cached document, if it's already been downloaded. Otherwise, this method resolves the hyperlink, downloads the target document, and displays the document in the appropriate application.

*expression*.**FollowHyperlink**(*Address*, *SubAddress*, *NewWindow*, *AddHistory*, *ExtraInfo*, *Method*, *HeaderInfo*)

*expression* Required. An expression that returns a **Workbook** object.

**Address** Required **String**. The address of the target document.

**SubAddress** Optional **Variant**. The location within the target document. The default value is the empty string.

**NewWindow** Optional **Variant**. **True** to display the target application in a new window. The default value is **False**.

**AddHistory** Optional **Variant**. Not used. Reserved for future use.

**ExtraInfo** Optional **Variant**. A **String** or byte array that specifies additional information for HTTP to use to resolve the hyperlink. For example, you can use **ExtraInfo** to specify the coordinates of an image map, the contents of a form, or a FAT file name.

**Method** Optional **Variant**. Specifies the way **ExtraInfo** is attached. Can be one of the following [MsoExtraInfoMethod](#) constants.

MsoExtraInfoMethod type can be one of these MsoExtraInfoMethod constants.

**msoMethodGet**. **ExtraInfo** is a **String** that's appended to the address.

**msoMethodPost**. **ExtraInfo** is posted as a **String** or byte array.

**HeaderInfo** Optional **Variant**. A **String** that specifies header information for the HTTP request. The default value is an empty string.

## Example

This example loads the document at [example.microsoft.com](http://example.microsoft.com) in a new window and adds it to the History folder.

```
ActiveWorkbook.FollowHyperlink Address:="http://example.microsoft.co  
NewWindow:=True
```



[Show All](#)

# Format Method

Sets a PivotTable report to one of the predefined indented, nonindented, or cross-tabulated formats.

*expression*.**Format**(*Format*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*Format* Required [XIPivotFormatType](#). Specifies the type of report formatting to be applied to the specified PivotTable report.

XIPivotFormatType can be one of these XIPivotFormatType constants.

**xIPTClassic**

**xIPTNone**

**xlReport1**

**xlReport10**

**xlReport2**

**xlReport3**

**xlReport4**

**xlReport5**

**xlReport6**

**xlReport7**

**xlReport8**

**xlReport9**

**xlTable1**

**xlTable10**

**xlTable2**

**xlTable3**

**xlTable4**

**xlTable5**

**xlTable6**

**xlTable7**

**xlTable8**

**xlTable9**

## Example

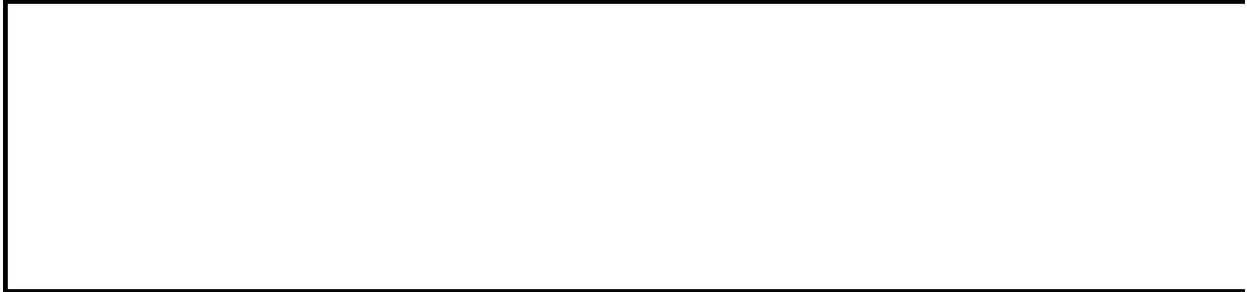
This example applies the xlReport4 indented format to the fourth PivotTable report on the active worksheet.

```
ActiveSheet.PivotTables("PivotTable4").Format xlReport4
```



# ForwardMailer Method

You have requested Help for a Visual Basic keyword used only on the Macintosh. For information about this keyword, consult the language reference Help included with Microsoft Office Macintosh Edition.



# FunctionWizard Method

Starts the Function Wizard for the upper-left cell of the range.

*expression*.**FunctionWizard**

*expression* Required. An expression that returns a **Range** object.

## Example

This example starts the Function Wizard for the active cell on Sheet1.

```
Worksheets("Sheet1").Activate  
ActiveCell.FunctionWizard
```



# GetChartElement Method

Returns information about the chart element at specified X and Y coordinates. This method is unusual in that you specify values for only the first two arguments. Microsoft Excel fills in the other arguments, and your code should examine those values when the method returns.

*expression*.**GetChartElement**(*X*, *Y*, *ElementID*, *Arg1*, *Arg2*)

*expression* Required. An expression that returns a **Chart** object.

*X* Required **Long**. The X coordinate of the chart element.

*Y* Required **Long**. The Y coordinate of the chart element.

*ElementID* Required **Long**. When the method returns, this argument contains the **XLChartItem** value of the chart element at the specified coordinates. For more information, see the “Remarks” section.

*Arg1* Required **Long**. When the method returns, this argument contains information related to the chart element. For more information, see the “Remarks” section.

*Arg2* Required **Long**. When the method returns, this argument contains information related to the chart element. For more information, see the “Remarks” section.

## Remarks

The value of *ElementID* after the method returns determines whether *Arg1* and *Arg2* contain any information, as shown in the following table.

<b>ElementID</b>	<b>Arg1</b>	<b>Arg2</b>
<b>xlAxis</b>	AxisIndex	AxisType
<b>xlAxisTitle</b>	AxisIndex	AxisType
<b>xlDisplayUnitLabel</b>	AxisIndex	AxisType
<b>xlMajorGridlines</b>	AxisIndex	AxisType
<b>xlMinorGridlines</b>	AxisIndex	AxisType
<b>xlPivotChartDropZone</b>	DropZoneType	None
<b>xlPivotChartFieldButton</b>	DropZoneType	PivotFieldIndex
<b>xlDownBars</b>	GroupIndex	None
<b>xlDropLines</b>	GroupIndex	None
<b>xlHiLoLines</b>	GroupIndex	None
<b>xlRadarAxisLabels</b>	GroupIndex	None
<b>xlSeriesLines</b>	GroupIndex	None
<b>xlUpBars</b>	GroupIndex	None
<b>xlChartArea</b>	None	None
<b>xlChartTitle</b>	None	None
<b>xlCorners</b>	None	None
<b>xlDataTable</b>	None	None
<b>xlFloor</b>	None	None
<b>xlLegend</b>	None	None
<b>xlNothing</b>	None	None
<b>xlPlotArea</b>	None	None
<b>xlWalls</b>	None	None
<b>xlDataLabel</b>	SeriesIndex	PointIndex
<b>xlErrorBars</b>	SeriesIndex	None
<b>xlLegendEntry</b>	SeriesIndex	None
<b>xlLegendKey</b>	SeriesIndex	None

<b>xlSeries</b>	SeriesIndex	PointIndex
<b>xlShape</b>	ShapeIndex	None
<b>xlTrendline</b>	SeriesIndex	TrendLineIndex
<b>xlXErrorBars</b>	SeriesIndex	None
<b>xlYErrorBars</b>	SeriesIndex	None

The following table describes the meaning of **Arg1** and **Arg2** after the method returns.

<b>Argument</b>	<b>Description</b>
AxisIndex	Specifies whether the axis is primary or secondary. Can be one of the following <b>XIAxisGroup</b> constants: <b>xlPrimary</b> or <b>xlSecondary</b> .
AxisType	Specifies the axis type. Can be one of the following <b>XIAxisType</b> constants: <b>xlCategory</b> , <b>xlSeriesAxis</b> , or <b>xlValue</b> .
DropZoneType	Specifies the drop zone type: column, data, page, or row field. Can be one of the following <b>XIPivotFieldOrientation</b> constants: <b>xlColumnField</b> , <b>xlDataField</b> , <b>xlPageField</b> , or <b>xlRowField</b> . The column and row field constants specify the series and category fields, respectively.
GroupIndex	Specifies the offset within the <a href="#">ChartGroups</a> collection for a specific chart group.
PivotFieldIndex	Specifies the offset within the <a href="#">PivotFields</a> collection for a specific column (series), data, page, or row (category) field. -1 if the drop zone type is <b>xlDataField</b> .
PointIndex	Specifies the offset within the <a href="#">Points</a> collection for a specific point within a series. A value of - 1 indicates that all data points are selected.
SeriesIndex	Specifies the offset within the <a href="#">Series</a> collection for a specific series.
ShapeIndex	Specifies the offset within the <a href="#">Shapes</a> collection for a specific shape.
TrendlineIndex	Specifies the offset within the <a href="#">Trendlines</a> collection for a specific trendline within a series.

## Example

This example warns the user if she moves the mouse over the chart legend.

```
Private Sub Chart_MouseMove(ByVal Button As Long, _  
    ByVal Shift As Long, ByVal X As Long, ByVal Y As Long)  
    Dim IDNum As Long  
    Dim a As Long  
    Dim b As Long  
  
    ActiveChart.GetChartElement X, Y, IDNum, a, b  
    If IDNum = xlLegendEntry Then _  
        MsgBox "WARNING: Move away from the legend"  
End Sub
```



# GetCustomListContents Method

Returns a custom list (an array of strings).

*expression*.**GetCustomListContents**(*ListNum*)

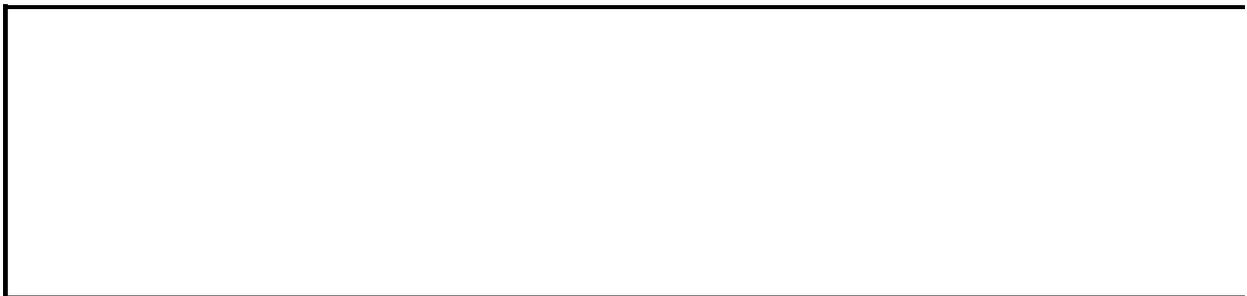
*expression* Required. An expression that returns an **Application** object.

*ListNum* Required **Long**. The list number.

## Example

This example writes the elements of the first custom list in column one on Sheet1.

```
listArray = Application.GetCustomListContents(1)
For i = LBound(listArray, 1) To UBound(listArray, 1)
    Worksheets("sheet1").Cells(i, 1).Value = listArray(i)
Next i
```



# GetCustomListNum Method

Returns the custom list number for an array of strings. You can use this method to match both built-in lists and custom-defined lists.

*expression*.**GetCustomListNum**(*ListArray*)

*expression* Required. An expression that returns an **Application** object.

*ListArray* Required **Variant**. An array of strings.

## Remarks

This method generates an error if there's no corresponding list.

## Example

This example deletes a custom list.

```
n = Application.GetCustomListNum(Array("cogs", "sprockets", _  
    "widgets", "gizmos"))  
Application.DeleteCustomList n
```



# GetData Method

*expression*.**GetData**(*Name*)

*expression* Required. An expression that returns a **PivotTable** object.

**Name** Required **String**. Describes a single cell in the PivotTable report, using syntax similar to the [PivotSelect](#) method or the PivotTable report references in calculated item formulas.

## Example

This example shows the sum of revenues for apples in January (Data field = Revenue, Product = Apples, Month = January).

```
Msgbox ActiveSheet.PivotTables(1) _  
    .GetData("'Sum of Revenue' Apples January")
```



# GetOpenFilename Method

Displays the standard **Open** dialog box and gets a file name from the user without actually opening any files.

*expression*.**GetOpenFilename**(*FileFilter*, *FilterIndex*, *Title*, *ButtonText*, *MultiSelect*)

*expression* Required. An expression that returns an **Application** object.

**FileFilter** Optional **Variant**. A string specifying file filtering criteria.

This string consists of pairs of file filter strings followed by the MS-DOS wildcard file filter specification, with each part and each pair separated by commas. Each separate pair is listed in the **Files of type** drop-down list box. For example, the following string specifies two file filters— text and addin: "Text Files (\*.txt),\*.txt,Add-In Files (\*.xla),\*.xla".

To use multiple MS-DOS wildcard expressions for a single file filter type, separate the wildcard expressions with semicolons; for example, "Visual Basic Files (\*.bas; \*.txt),\*.bas;\*.txt".

If omitted, this argument defaults to "All Files (\*.\*),\*.\*".

**FilterIndex** Optional **Variant**. Specifies the index numbers of the default file filtering criteria, from 1 to the number of filters specified in **FileFilter**. If this argument is omitted or greater than the number of filters present, the first file filter is used.

**Title** Optional **Variant**. Specifies the title of the dialog box. If this argument is omitted, the title is "Open."

**ButtonText** Optional **Variant**. Macintosh only.

**MultiSelect** Optional **Variant**. **True** to allow multiple file names to be selected. **False** to allow only one file name to be selected. The default value is **False**

## Remarks

This method returns the selected file name or the name entered by the user. The returned name may include a path specification. If ***MultiSelect*** is **True**, the return value is an array of the selected file names (even if only one filename is selected). Returns **False** if the user cancels the dialog box.

This method may change the current drive or folder.

## Example

This example displays the **Open** dialog box, with the file filter set to text files. If the user chooses a file name, the code displays that file name in a message box.

```
fileToOpen = Application _  
    .GetOpenFilename("Text Files (*.txt), *.txt")  
If fileToOpen <> False Then  
    MsgBox "Open " & fileToOpen  
End If
```



# GetPhonetic Method

Returns the Japanese phonetic text of the specified text string. This method is available to you only if you have selected or installed Japanese language support for Microsoft Office.

*expression*.**GetPhonetic**(*Text*)

*expression* An expression that returns an **Application** object.

**Text** Optional **Variant**. Specifies the text to be converted to phonetic text. If you omit this argument, the next possible phonetic text string (if any) of the previously specified **Text** is returned. If there are no more possible phonetic text strings, an empty string is returned.

## Example

This example displays all of the possible phonetic text strings from the specified string.

```
strPhoText = Application.GetPhonetic("純子")  
While strPhoText <> ""  
    MsgBox strPhoText  
    strPhoText = Application.GetPhonetic()  
Wend
```



# GetPivotData Method

Returns a **Range** object with information about a data item in a PivotTable report.

*expression*.**GetPivotData**(*DataField*, *Field1*, *Item1*, *Field2*, *Item2*, *Field3*, *Item3*, *Field4*, *Item4*, *Field5*, *Item5*, *Field6*, *Item6*, *Field7*, *Item7*, *Field8*, *Item8*, *Field9*, *Item9*, *Field10*, *Item10*, *Field11*, *Item11*, *Field12*, *Item12*, *Field13*, *Item13*, *Field14*, *Item14*, *Field15*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**DataField** Optional **Variant**. The name of the field containing the data for the PivotTable.

**Field1** Optional **Variant**. The name of a column or row field in the PivotTable report.

**Item1** Optional **Variant**. The name of an item in **Field1**.

**Field2** Optional **Variant**. The name of a column or row field in the PivotTable report.

**Item2** Optional **Variant**. The name of an item in **Field2**.

**Field3** Optional **Variant**. The name of a column or row field in the PivotTable report.

**Item3** Optional **Variant**. The name of an item in **Field3**.

**Field4** Optional **Variant**. The name of a column or row field in the PivotTable report.

**Item4** Optional **Variant**. The name of an item in **Field4**.

**Field5** Optional **Variant**. The name of a column or row field in the PivotTable report.

**Item5** Optional **Variant**. The name of an item in **Field5**.

**Field6** Optional **Variant**. The name of a column or row field in the PivotTable report.

**Item6** Optional **Variant**. The name of an item in **Field6**.

**Field7** Optional **Variant**. The name of a column or row field in the PivotTable report.

**Item7** Optional **Variant**. The name of an item in **Field7**.

**Field8** Optional **Variant**. The name of a column or row field in the PivotTable report.

**Item8** Optional **Variant**. The name of an item in **Field8**.

**Field9** Optional **Variant**. The name of a column or row field in the PivotTable report.

**Item9** Optional **Variant**. The name of an item in **Field9**.

**Field10** Optional **Variant**. The name of a column or row field in the PivotTable report.

**Item10** Optional **Variant**. The name of an item in **Field10**.

**Field11** Optional **Variant**. The name of a column or row field in the PivotTable report.

**Item11** Optional **Variant**. The name of an item in **Field11**.

**Field12** Optional **Variant**. The name of a column or row field in the PivotTable report.

**Item12** Optional **Variant**. The name of an item in **Field12**.

**Field13** Optional **Variant**. The name of a column or row field in the PivotTable report.

**Item13** Optional **Variant**. The name of an item in **Field13**.

**Field14** Optional **Variant**. The name of a column or row field in the PivotTable report.

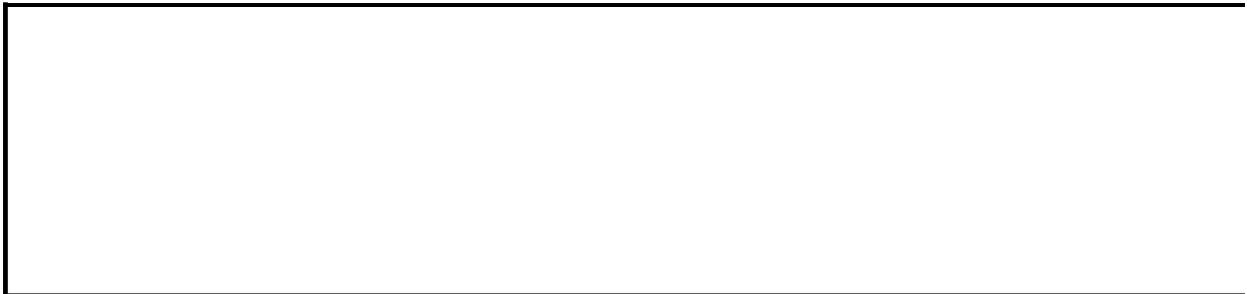
**Item14** Optional **Variant**. The name of an item in **Field14**.

**Field15** Optional **Variant**. The name of a column or row field in the PivotTable report.

## Example

In this example, Microsoft Excel returns the quantity of chairs in the warehouse to the user. This example assumes a PivotTable report exists on the active worksheet. Also, this example assumes that, in the report, the title of the data field is "Quantity", a field titled "Warehouse" exists, and a data item titled "Chairs" exists in the Warehouse field.

```
Sub UseGetPivotData()  
  
    Dim rngTableItem As Range  
  
    ' Get PivotData for the quantity of chairs in the warehouse.  
    Set rngTableItem = ActiveCell. _  
        PivotTable.GetPivotData("Quantity", "Warehouse", "Chairs")  
  
    MsgBox "The quantity of chairs in the warehouse is: " & rngTable  
  
End Sub
```



# GetSaveAsFilename Method

Displays the standard **Save As** dialog box and gets a file name from the user without actually saving any files.

*expression*.**GetSaveAsFilename**(*InitialFilename*, *FileFilter*, *FilterIndex*, *Title*, *ButtonText*)

*expression* Required. An expression that returns an **Application** object.

**InitialFilename** Optional **Variant**. Specifies the suggested file name. If this argument is omitted, Microsoft Excel uses the active workbook's name.

**FileFilter** Optional **Variant**. A string specifying file filtering criteria.

This string consists of pairs of file filter strings followed by the MS-DOS wildcard file filter specification, with each part and each pair separated by commas. Each separate pair is listed in the **Files of type** drop-down list box. For example, the following string specifies two file filters, text and addin: "Text Files (\*.txt), \*.txt, Add-In Files (\*.xla), \*.xla".

To use multiple MS-DOS wildcard expressions for a single file filter type, separate the wildcard expressions with semicolons; for example, "Visual Basic Files (\*.bas; \*.txt),\*.bas;\*.txt".

If omitted, this argument defaults to "All Files (\*.\*),\*.\*".

**FilterIndex** Optional **Variant**. Specifies the index number of the default file filtering criteria, from 1 to the number of filters specified in **FileFilter**. If this argument is omitted or greater than the number of filters present, the first file filter is used.

**Title** Optional **Variant**. Specifies the title of the dialog box. If this argument is omitted, the default title is used.

**ButtonText** Optional **Variant**. Macintosh only.

## Remarks

This method returns the selected file name or the name entered by the user. The returned name may include a path specification. Returns **False** if the user cancels the dialog box.

This method may change the current drive or folder.

## Example

This example displays the **Save As** dialog box, with the file filter set to text files. If the user chooses a file name, the example displays that file name in a message box.

```
fileSaveName = Application.GetSaveAsFilename( _  
    fileFilter:="Text Files (*.txt), *.txt")  
If fileSaveName <> False Then  
    MsgBox "Save as " & fileSaveName  
End If
```



# GoalSeek Method

Calculates the values necessary to achieve a specific goal. If the goal is an amount returned by a formula, this calculates a value that, when supplied to your formula, causes the formula to return the number you want. Returns **True** if the goal seek is successful.

*expression*.**GoalSeek**(*Goal*, *ChangingCell*)

*expression* Required. An expression that returns a **Range** object. Must be a single cell.

**Goal** Required **Variant**. The value you want returned in this cell.

**ChangingCell** Required **Range**. Specifies which cell should be changed to achieve the target value.

## Example

This example assumes that Sheet1 has a cell named "Polynomial" that contains the formula  $=(X^3)+(3*X^2)+6$  and another cell named "X" that's empty. The example finds a value for X so that Polynomial contains the value 15.

```
Worksheets("Sheet1").Range("Polynomial").GoalSeek _  
    Goal:=15, _  
    ChangingCell:=Worksheets("Sheet1").Range("X")
```



# Goto Method

Selects any range or Visual Basic procedure in any workbook, and activates that workbook if it's not already active.

*expression*.**Goto**(*Reference*, *Scroll*)

*expression* Required. An expression that returns an **Application** object.

**Reference** Optional **Variant**. The destination. Can be a **Range** object, a string that contains a cell reference in R1C1-style notation, or a string that contains a Visual Basic procedure name. If this argument is omitted, the destination is the last range you used the **Goto** method to select.

**Scroll** Optional **Variant**. **True** to scroll through the window so that the upper-left corner of the range appears in the upper-left corner of the window. **False** to not scroll through the window. The default is **False**.

## Remarks

This method differs from the [Select](#) method in the following ways:

- If you specify a range on a sheet that's not on top, Microsoft Excel will switch to that sheet before selecting. (If you use **Select** with a range on a sheet that's not on top, the range will be selected but the sheet won't be activated).
- This method has a **Scroll** argument that lets you scroll through the destination window.
- When you use the **Goto** method, the previous selection (before the **Goto** method runs) is added to the array of previous selections (for more information, see the [PreviousSelections](#) property). You can use this feature to quickly jump between as many as four selections.
- The **Select** method has a **Replace** argument; the **Goto** method doesn't.

## Example

This example selects cell A154 on Sheet1 and then scrolls through the worksheet to display the range.

```
Application.Goto Reference:=Worksheets("Sheet1").Range("A154"), _  
    scroll:=True
```



[Show All](#)

# Group Method

 [Group method as it applies to the \*\*ShapeRange\*\* object.](#)

Groups the shapes in the specified range. Returns the grouped shapes as a single **Shape** object.

*expression*.**Group**

*expression* Required. An expression that returns a **ShapeRange** object.

 [Group method as it applies to the \*\*Range\*\* object.](#)

When the **Range** object represents a single cell in a PivotTable field's data range, the **Group** method performs numeric or date-based grouping in that field.

*expression*.**Group**(*Start*, *End*, *By*, *Periods*)

*expression* Required. An expression that returns a **Range** object.

**Start** Optional **VARIANT**. The first value to be grouped. If this argument is omitted or **True**, the first value in the field is used.

**End** Optional **VARIANT**. The last value to be grouped. If this argument is omitted or **True**, the last value in the field is used.

**By** Optional **VARIANT**. If the field is numeric, this argument specifies the size of each group. If the field is a date, this argument specifies the number of days in each group if element 4 in the **Periods** array is **True** and all the other elements are **False**. Otherwise, this argument is ignored. If this argument is omitted, Microsoft Excel automatically chooses a default group size.

**Periods** Optional **VARIANT**. An array of Boolean values that specify the period for the group, as shown in the following table.

Array element	Period
---------------	--------

1	Seconds
---	---------

2	Minutes
3	Hours
4	Days
5	Months
6	Quarters
7	Years

If an element in the array is **True**, a group is created for the corresponding time; if the element is **False**, no group is created. If the field isn't a date field, this argument is ignored.

## Remarks

Because a group of shapes is treated as a single shape, grouping and ungrouping shapes changes the number of items in the **Shapes** collection and changes the index numbers of items that come after the affected items in the collection.

The **Range** object must be a single cell in the PivotTable field's data range. If you attempt to apply this method to more than one cell, it will fail (without displaying an error message).

## Example

This example groups the field named ORDER\_DATE by 10-day periods.

```
Set pvtTable = Worksheets("Sheet1").Range("A3").PivotTable
Set groupRange = pvtTable.PivotFields("ORDER_DATE").DataRange
groupRange.Cells(1).Group by:=10, _
    periods:=Array(False, False, False, _
        True, False, False, False)
```



# Heartbeat Method

Determines if the real-time data server is still active. Returns a **Long** value. Zero or a negative number indicates failure; a positive number indicates that the server is active.

*expression*.**Heartbeat**

*expression* Required. An expression that returns an **IRtdServer** object.

## Remarks

The **Heartbeat** method is called by Microsoft Excel if the [HeartbeatInterval](#) property has elapsed since the last time Excel was called with the [UpdateNotify](#) method.

---

# Help Method

Displays a Help topic.

*expression*.**Help**(*HelpFile*, *HelpContextID*)

*expression* Required. An expression that returns an **Application** object.

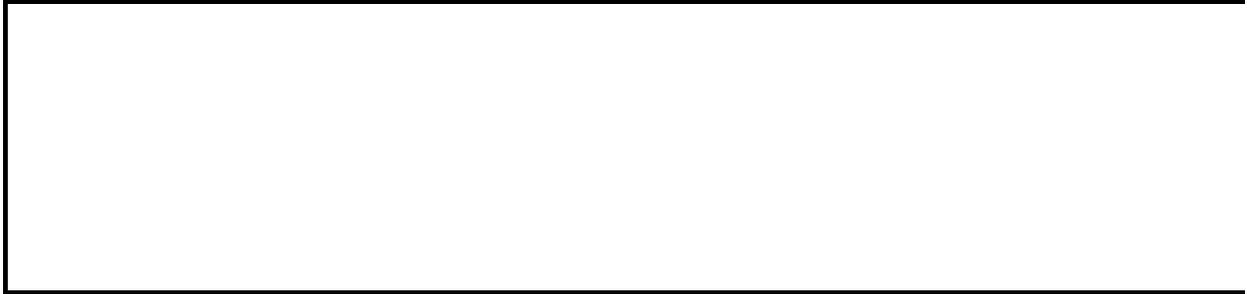
**helpFile** Optional **Variant**. The name of the online Help file you want to display. If this argument isn't specified, Microsoft Excel Help is used.

**helpContextID** Optional **Variant**. Specifies the context ID number for the Help topic. If this argument isn't specified, the **Help Topics** dialog box is displayed.

## Example

This example displays topic number 65527 in the Help file Otisapp.hlp.

```
Application.Help "OTISAPP.HLP", 65527
```



# HighlightChangesOptions Method

Controls how changes are shown in a shared workbook.

*expression*.**HighlightChangesOptions**(*When, Who, Where*)

*expression* Required. An expression that returns a **Workbook** object.

**When** Optional **Variant**. The changes that are shown. Can be one of the following **XlHighlightChangesTime** constants: **xlSinceMyLastSave**, **xlAllChanges**, or **xlNotYetReviewed**.

**Who** Optional **Variant**. The user or users whose changes are shown. Can be "Everyone," "Everyone but Me," or the name of one of the users of the shared workbook.

**Where** Optional **Variant**. An A1-style range reference that specifies the area to check for changes.

## Example

This example shows changes to the shared workbook on a separate worksheet.

```
With ActiveWorkbook
    .HighlightChangesOptions _
        When:=xlSinceMyLastSave, _
        Who:="Everyone"
    .ListChangesOnNewSheet = True
End With
```



[Show All](#)

# Import Method

 [Import method as it applies to the \*\*SoundNote\*\* object.](#)

This method should not be used. Sound notes have been removed from Microsoft Excel.

 [Import method as it applies to the \*\*XmlMap\*\* object.](#)

Imports data from the specified XML data file into cells that have been mapped to the specified **XmlMap** object. Returns [\*\*XlXmlImportResult\*\*](#).

**XlXmlImportResult** can be one of the following **XlXmlImportResult** constants  
**xlXmlImportElementsTruncated** The contents of the specified XML data file have been truncated because the XML data file is too large for a cell.

**xlXmlImportSuccess** The XML data file was successfully imported.

**xlXmlImportValidationFailed** The contents of the XML data file do not match the specified schema map.

*expression*.**Import**(*Url*, *Overwrite*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Url** Required **String**. The path to the XML data to import. The path can be specified in Universal Naming convention (UNC) or Uniform Resource Locator (URL) format. The file can be an XML data file or a valid Office XML data Source Connection (.uxdc) file.

**Overwrite** Optional **Variant**. Set to **True** to overwrite existing data. Set to **False** to append to existing data. The default value is **False**.



[Show All](#)

# ImportXml Method

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Imports XML data from a **String** variable into cells that have been mapped to the specified [XmlMap](#) object. Returns an [XlXmlImportResult](#) constant.

**XlXmlImportResult** can be one of the following **XlXmlImportResult** constants:

**xlXmlImportElementsTruncated** The contents of the specified XML data file have been truncated because some of the XML data is too large for a cell.

**xlXmlImportSuccess** The XML data file was successfully imported.

**xlXmlImportValidationFailed** The contents of the XML data file do not match the specified schema map.

*expression.ImportXml(XmlData, Overwrite)*

*expression* Required. An expression that returns an [XmlMap](#) object.

**XmlData** Required **String**. The string that contains the XML data to import.

**Overwrite** Optional **Boolean** value. Specifies whether to overwrite the contents of cells that are currently mapped to the specified XML map. Set to **True** to overwrite the cells; set to **False** to append the data to the existing range. If this parameter is not specified, the current value of the [AppendOnImport](#) property of the XML map determines whether the contents of cells are overwritten or not.

## Remarks

To import the contents of an XML data file into cells mapped to a specific schema map, use the [Import](#) method of the **XmlMap** object.

---

---

---

[Show All](#)

# InchesToPoints Method

Converts a measurement from inches to [points](#).

*expression*.**InchesToPoints**(*Inches*)

*expression* Required. An expression that returns an **Application** object.

*Inches* Required **Double**. Specifies the inch value to be converted to points.

## Example

This example sets the left margin of Sheet1 to 2.5 inches.

```
Worksheets("Sheet1").PageSetup.LeftMargin = _  
Application.InchesToPoints(2.5)
```



# IncrementBrightness Method

Changes the brightness of the picture by the specified amount. Use the [Brightness](#) property to set the absolute brightness of the picture.

*expression*.IncrementBrightness(*Increment*)

*expression* Required. An expression that returns a **PictureFormat** object.

**Increment** Required **Single**. Specifies how much to change the value of the **Brightness** property for the picture. A positive value makes the picture brighter; a negative value makes the picture darker.

## Remarks

You cannot adjust the brightness of a picture past the upper or lower limit for the **Brightness** property. For example, if the **Brightness** property is initially set to 0.9 and you specify 0.3 for the *Increment* argument, the resulting brightness level will be 1.0, which is the upper limit for the **Brightness** property, instead of 1.2.

## Example

This example creates a duplicate of shape one on myDocument and then moves and darkens the duplicate. For the example to work, shape one must be either a picture or an OLE object.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(1).Duplicate
    .PictureFormat.IncrementBrightness -0.2
    .IncrementLeft 50
    .IncrementTop 50
End With
```



# IncrementContrast Method

Changes the contrast of the picture by the specified amount. Use the [Contrast](#) property to set the absolute contrast for the picture.

*expression*.**IncrementContrast**(*Increment*)

*expression* Required. An expression that returns a **PictureFormat** object.

**Increment** Required **Single**. Specifies how much to change the value of the **Contrast** property for the picture. A positive value increases the contrast; a negative value decreases the contrast.

## Remarks

You cannot adjust the contrast of a picture past the upper or lower limit for the **Contrast** property. For example, if the **Contrast** property is initially set to 0.9 and you specify 0.3 for the **Increment** argument, the resulting contrast level will be 1.0, which is the upper limit for the **Contrast** property, instead of 1.2.

## Example

This example increases the contrast for all pictures on myDocument that aren't already set to maximum contrast.

```
Set myDocument = Worksheets(1)
For Each s In myDocument.Shapes
    If s.Type = msoPicture Or s.Type = msoLinkedPicture Then
        s.PictureFormat.IncrementContrast 0.1
    End If
Next
```



# IncrementLeft Method

Moves the specified shape horizontally by the specified number of points.

*expression*.**IncrementLeft**(*Increment*)

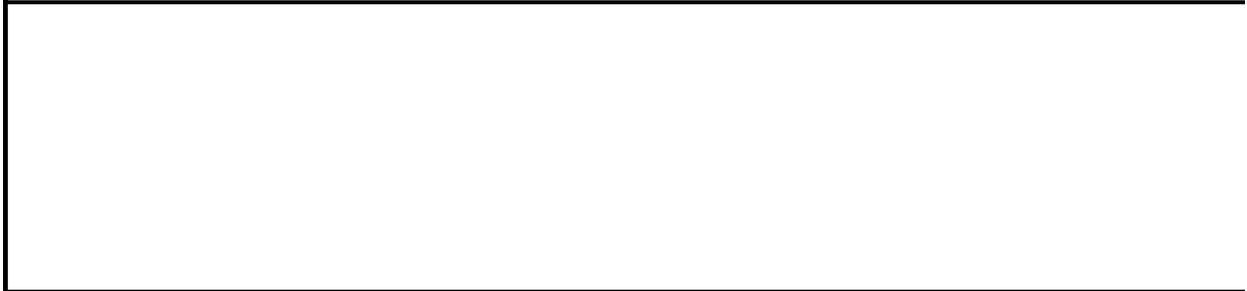
*expression* Required. An expression that returns a **Shape** object.

**Increment** Required **Single**. Specifies how far the shape is to be moved horizontally, in points. A positive value moves the shape to the right; a negative value moves it to the left.

## Example

This example duplicates shape one on myDocument, sets the fill for the duplicate, moves it 70 points to the right and 50 points up, and rotates it 30 degrees clockwise.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(1).Duplicate
    .Fill.PresetTextured msoTextureGranite
    .IncrementLeft 70
    .IncrementTop -50
    .IncrementRotation 30
End With
```



# IncrementOffsetX Method

Changes the horizontal offset of the shadow by the specified number of points. Use the [OffsetX](#) property to set the absolute horizontal shadow offset.

*expression*.IncrementOffsetX(*Increment*)

*expression* Required. An expression that returns a **ShadowFormat** object.

**Increment** Required **Single**. Specifies how far the shadow offset is to be moved horizontally, in points. A positive value moves the shadow to the right; a negative value moves it to the left.

## Example

This example moves the shadow on shape three on myDocument to the left by 3 points.

```
Set myDocument = Worksheets(1)  
myDocument.Shapes(3).Shadow.IncrementOffsetX -3
```



# IncrementOffsetY Method

Changes the vertical offset of the shadow by the specified number of points. Use the [OffsetY](#) property to set the absolute vertical shadow offset.

*expression*.IncrementOffsetY(*Increment*)

*expression* Required. An expression that returns a **ShadowFormat** object.

**Increment** Required **Single**. Specifies how far the shadow offset is to be moved vertically, in points. A positive value moves the shadow down; a negative value moves it up.

## Example

This example moves the shadow on shape three on myDocument up by 3 points.

```
Set myDocument = Worksheets(1)  
myDocument.Shapes(3).Shadow.IncrementOffsetY -3
```



# IncrementRotation Method

Changes the rotation of the specified shape around the z-axis by the specified number of degrees. Use the [Rotation](#) property to set the absolute rotation of the shape.

*expression*.IncrementRotation(*Increment*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Increment** Required **Single**. Specifies how far the shape is to be rotated horizontally, in degrees. A positive value rotates the shape clockwise; a negative value rotates it counterclockwise.

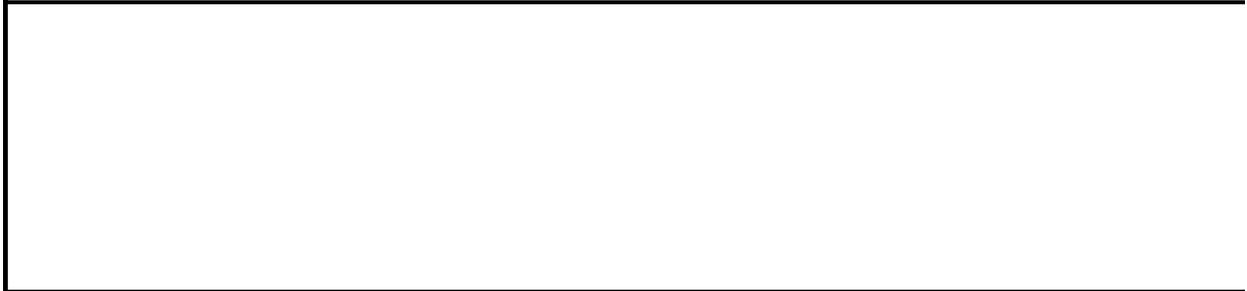
## Remarks

To rotate a three-dimensional shape around the x-axis or the y-axis, use the [IncrementRotationX](#) method or the [IncrementRotationY](#) method.

## Example

This example duplicates shape one on myDocument, sets the fill for the duplicate, moves it 70 points to the right and 50 points up, and rotates it 30 degrees clockwise.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(1).Duplicate
    .Fill.PresetTextured msoTextureGranite
    .IncrementLeft 70
    .IncrementTop -50
    .IncrementRotation 30
End With
```



# IncrementRotationX Method

Changes the rotation of the specified shape around the x-axis by the specified number of degrees. Use the [RotationX](#) property to set the absolute rotation of the shape around the x-axis.

*expression*.IncrementRotationX(*Increment*)

*expression* Required. An expression that returns a **ThreeDFormat** object.

**Increment** Required **Single**. Specifies how much (in degrees) the rotation of the shape around the x-axis is to be changed. Can be a value from – 90 through 90. A positive value tilts the shape up; a negative value tilts it down.

## Remarks

You cannot adjust the specified shape's rotation around the x-axis past the upper or lower limit for the **RotationX** property (90 degrees to – 90 degrees). For example, if the **RotationX** property is initially set to 80 and you specify 40 for the *Increment* argument, the resulting rotation will be 90 (the upper limit for the **RotationX** property) instead of 120.

To change the rotation of a shape around the y-axis, use the [IncrementRotationY](#) method. To change the rotation around the z-axis, use the [IncrementRotation](#) method.

## Example

This example tilts shape one on myDocument up 10 degrees. Shape one must be an extruded shape for you to see the effect of this code.

```
Set myDocument = Worksheets(1)  
myDocument.Shapes(1).ThreeD.IncrementRotationX 10
```



# IncrementRotationY Method

Changes the rotation of the specified shape around the y-axis by the specified number of degrees. Use the [RotationY](#) property to set the absolute rotation of the shape around the y-axis.

*expression*.IncrementRotationY(*Increment*)

*expression* Required. An expression that returns a **ThreeDFormat** object.

**Increment** Required **Single**. Specifies how much (in degrees) the rotation of the shape around the y-axis is to be changed. Can be a value from – 90 through 90. A positive value tilts the shape to the left; a negative value tilts it to the right.

## Remarks

To change the rotation of a shape around the x-axis, use the [IncrementRotationX](#) method. To change the rotation around the z-axis, use the [IncrementRotation](#) method.

You cannot adjust the specified shape's rotation around the y-axis shape past the upper or lower limit for the **RotationY** property (90 degrees to – 90 degrees). For example, if the **RotationY** property is initially set to 80 and you specify 40 for the *Increment* argument, the resulting rotation will be 90 (the upper limit for the **RotationY** property) instead of 120.

## Example

This example tilts shape one on myDocument 10 degrees to the right. Shape one must be an extruded shape for you to see the effect of this code.

```
Set myDocument = Worksheets(1)  
myDocument.Shapes(1).ThreeD.IncrementRotationY -10
```



# IncrementTop Method

Moves the specified shape vertically by the specified number of points.

*expression*.**IncrementTop**(*Increment*)

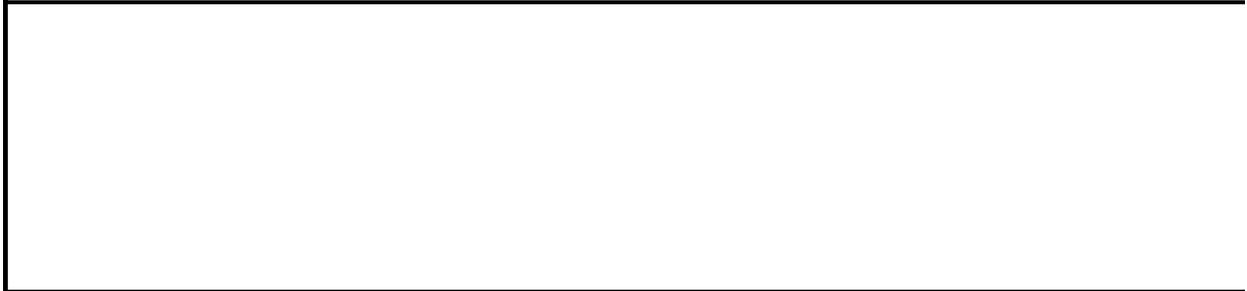
*expression* Required. An expression that returns a **Shape** object.

**Increment** Required **Single**. Specifies how far the shape object is to be moved vertically, in points. A positive value moves the shape down; a negative value moves it up.

## Example

This example duplicates shape one on myDocument, sets the fill for the duplicate, moves it 70 points to the right and 50 points up, and rotates it 30 degrees clockwise.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(1).Duplicate
    .Fill.PresetTextured msoTextureGranite
    .IncrementLeft 70
    .IncrementTop -50
    .IncrementRotation 30
End With
```



[Show All](#)

# InputBox Method

Displays a dialog box for user input. Returns the information entered in the dialog box.

*expression*.**InputBox**(**Prompt**, **Title**, **Default**, **Left**, **Top**, **HelpFile**, **HelpContextId**, **Type**)

*expression* Required. An expression that returns an **Application** object.

**Prompt** Required **String**. The message to be displayed in the dialog box. This can be a string, a number, a date, or a Boolean value (Microsoft Excel automatically coerces the value to a **String** before it's displayed).

**Title** Optional **VARIANT**. The title for the input box. If this argument is omitted, the default title is "Input."

**Default** Optional **VARIANT**. Specifies a value that will appear in the text box when the dialog box is initially displayed. If this argument is omitted, the text box is left empty. This value can be a **Range** object.

**Left** Optional **VARIANT**. Specifies an x position for the dialog box in relation to the upper-left corner of the screen, in **points**.

**Top** Optional **VARIANT**. Specifies a y position for the dialog box in relation to the upper-left corner of the screen, in points.

**HelpFile** Optional **VARIANT**. The name of the Help file for this input box. If the **HelpFile** and **HelpContextID** arguments are present, a Help button will appear in the dialog box.

**HelpContextId** Optional **VARIANT**. The context ID number of the Help topic in **HelpFile**.

**Type** Optional **VARIANT**. Specifies the return data type. If this argument is omitted, the dialog box returns text. Can be one or a sum of the following values.

Value	Meaning
-------	---------

- 0 A formula
- 1 A number
- 2 Text (a string)
- 4 A logical value (**True** or **False**)
- 8 A cell reference, as a **Range** object
- 16 An error value, such as #N/A
- 64 An array of values

You can use the sum of the allowable values for *Type*. For example, for an input box that can accept both text and numbers, set *Type* to 1 + 2.

## Remarks

Use **InputBox** to display a simple dialog box so that you can enter information to be used in a macro. The dialog box has an **OK** button and a **Cancel** button. If you choose the **OK** button, **InputBox** returns the value entered in the dialog box. If you click the **Cancel** button, **InputBox** returns **False**.

If **Type** is 0, **InputBox** returns the formula in the form of text— for example, " $=2*PI()/360$ ". If there are any references in the formula, they are returned as A1-style references. (Use [ConvertFormula](#) to convert between reference styles.)

If **Type** is 8, **InputBox** returns a **Range** object. You must use the **Set** statement to assign the result to a **Range** object, as shown in the following example.

```
Set myRange = Application.InputBox(prompt := "Sample", type := 8)
```

If you don't use the **Set** statement, the variable is set to the value in the range, rather than the **Range** object itself.

If you use the **InputBox** method to ask the user for a formula, you must use the [FormulaLocal](#) property to assign the formula to a **Range** object. The input formula will be in the user's language.

The **InputBox** method differs from the **InputBox** function in that it allows selective validation of the user's input, and it can be used with Microsoft Excel objects, error values, and formulas. Note that `Application.InputBox` calls the **InputBox** method; `InputBox` with no object qualifier calls the **InputBox** function.

## Example

This example prompts the user for a number.

```
myNum = Application.InputBox("Enter a number")
```

This example prompts the user to select a cell on Sheet1. The example uses the *Type* argument to ensure that the return value is a valid cell reference (a **Range** object).

```
Worksheets("Sheet1").Activate  
Set myCell = Application.InputBox( _  
    prompt:="Select a cell", Type:=8)
```



[Show All](#)

# Insert Method

 [Insert method as it applies to the \*\*Range\*\* object.](#)

Inserts a cell or a range of cells into the worksheet or macro sheet and shifts other cells away to make space.

*expression.Insert(Shift, CopyOrigin)*

*expression* Required. An expression that returns a **Range** object.

**Shift** Optional **Variant**. Specifies which way to shift the cells. Can be one of the following **XlInsertShiftDirection** constants: **xlShiftToRight** or **xlShiftDown**. If this argument is omitted, Microsoft Excel decides based on the shape of the range.

**CopyOrigin** Optional **Variant**. The copy origin.

 [Insert method as it applies to the \*\*Characters\*\* object.](#)

Inserts a string preceding the selected characters.

*expression.Insert(String)*

*expression* Required. An expression that returns a **Characters** object.

**String** Required **String**. The string to insert.

 [Insert method as it applies to the \*\*ShapeNodes\*\* object.](#)

Inserts a node into a freeform shape.

*expression.Insert(Index, SegmentType, EditingType, X1, Y1, X2, Y2, X3, Y3)*

*expression* Required. An expression that returns a **ShapeNodes** object.

**Index** Required **Long**. The number of the shape node after which to insert a

new node.

**SegmentType** Required [MsoSegmentType](#). The segment type.

MsoSegmentType can be one of these MsoSegmentType constants.

**msoSegmentCurve**

**msoSegmentLine**

**EditingType** Required [MsoEditingType](#). The editing type.

MsoEditingType can be one of these MsoEditingType constants.

**msoEditingAuto**

**msoEditingCorner**

**msoEditingSmooth**

**msoEditingSymmetric**

**X1** Required **Single**. If the **EditingType** of the new segment is **msoEditingAuto**, this argument specifies the horizontal distance, measured in points, from the upper-left corner of the document to the end point of the new segment. If the **EditingType** of the new node is **msoEditingCorner**, this argument specifies the horizontal distance, measured in points, from the upper-left corner of the document to the first control point for the new segment.

**Y1** Required **Single**. If the **EditingType** of the new segment is **msoEditingAuto**, this argument specifies the vertical distance, measured in points, from the upper-left corner of the document to the end point of the new segment. If the **EditingType** of the new node is **msoEditingCorner**, this argument specifies the vertical distance, measured in points, from the upper-left corner of the document to the first control point for the new segment.

**X2** Optional **Single**. If the **EditingType** of the new segment is **msoEditingCorner**, this argument specifies the horizontal distance, measured in points, from the upper-left corner of the document to the second control point for the new segment. If the **EditingType** of the new segment is **msoEditingAuto**, don't specify a value for this argument.

**Y2** Optional **Single**. If the **EditingType** of the new segment is **msoEditingCorner**, this argument specifies the vertical distance, measured in

points, from the upper-left corner of the document to the second control point for the new segment. If the *EditingType* of the new segment is **msoEditingAuto**, don't specify a value for this argument.

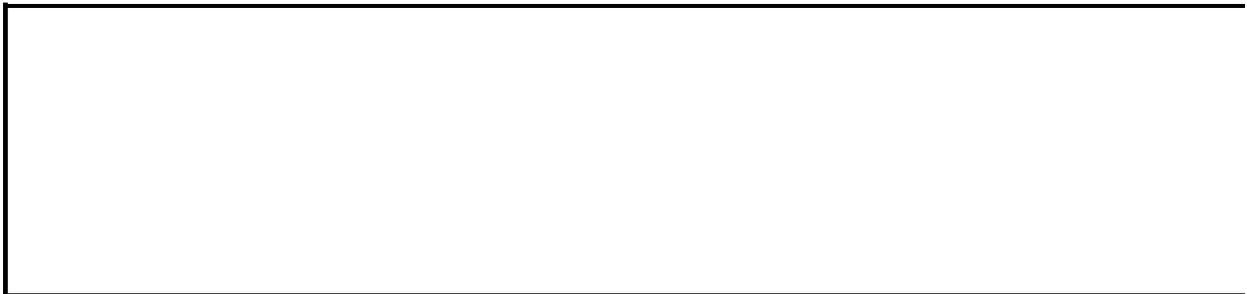
**X3** Optional **Single**. If the *EditingType* of the new segment is **msoEditingCorner**, this argument specifies the horizontal distance, measured in points, from the upper-left corner of the document to the end point of the new segment. If the *EditingType* of the new segment is **msoEditingAuto**, don't specify a value for this argument.

**Y3** Optional **Single**. If the *EditingType* of the new segment is **msoEditingCorner**, this argument specifies the vertical distance, measured in points, from the upper-left corner of the document to the end point of the new segment. If the *EditingType* of the new segment is **msoEditingAuto**, don't specify a value for this argument.

## Example

This example selects the third shape in the active document, checks whether the shape is a **Freeform** object, and if it is, inserts a node. This example assumes three shapes exist on the active worksheet.

```
Sub InsertShapeNode()  
    ActiveSheet.Shapes(3).Select  
    With Selection.ShapeRange  
        If .Type = msoFreeform Then  
            .Nodes.Insert _  
                Index:=3, SegmentType:=msoSegmentCurve, _  
                EditingType:=msoEditingSymmetric, X1:=35, Y1:=100  
            .Fill.ForeColor.RGB = RGB(0, 0, 200)  
            .Fill.Visible = msoTrue  
        Else  
            MsgBox "This shape is not a Freeform object."  
        End If  
    End With  
End Sub
```



# InsertIndent Method

Adds an indent to the specified range.

*expression*.**InsertIndent**(*InsertAmount*)

*expression* Required. An expression that returns a **Range** object.

*InsertAmount* Required **Long**. The amount to be added to the current indent.

## Remarks

Using this method to set the indent level to a number less than 0 (zero) or greater than 15 causes an error.

Use the **IndentLevel** property to return the indent level for a range.

## Example

This example decreases the indent level in cell A10.

```
With Range("a10")  
    .InsertIndent -1  
End With
```



[Show All](#)

# InstallManifest Method

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Installs the specified [XML expansion pack](#) on the user's computer, making an XML smart document solution available to one or more users.

*expression*.**InstallManifest**(*Path*, *InstallForAllUsers*)

*expression* Required. An expression that returns an [XmlNamespaces](#) collection.

*Path* Required **String**. The path and file name of the XML expansion pack.

*InstallForAllUsers* Optional **Boolean**. **True** installs the XML expansion pack and makes it available to all users on a machine. **False** makes the XML expansion pack available for the current user only. Default is **False**.

## Remarks

For security purposes, you cannot install an unsigned manifest. For more information about manifests, see the Smart Document Software Development Kit (SDK) on the [Microsoft Developer Network \(MSDN\)](#) Web site.

## Example

The following example installs the SimpleSample smart document solution on the user's computer and makes it available only to the current user.

```
Application.XMLNamespaces.InstallManifest _  
    "http://smartdocuments/simplesample/manifest.xml"
```

**Note** The SimpleSample schema is included in the Smart Document Software Development Kit (SDK). For more information, see the Smart Document SDK.



# Intersect Method

Returns a [Range](#) object that represents the rectangular intersection of two or more ranges.

*expression*.**Intersect**(*Arg1*, *Arg2*, ...)

*expression* Optional. An expression that returns an **Application** object.

*Arg1*, *Arg2*, ... Required **Range**. The intersecting ranges. At least two **Range** objects must be specified.

## Example

This example selects the intersection of two named ranges, rg1 and rg2, on Sheet1. If the ranges don't intersect, the example displays a message.

```
Worksheets("Sheet1").Activate
Set isect = Application.Intersect(Range("rg1"), Range("rg2"))
If isect Is Nothing Then
    MsgBox "Ranges do not intersect"
Else
    isect.Select
End If
```



[Show All](#)

# Item Method

 [Item method as it applies to the \*\*Axes\*\* object.](#)

Returns a single [Axis](#) object from an **Axes** collection.

*expression*.**Item**(*Type*, *AxisGroup*)

*expression* Required. An expression that returns an **Axes** collection.

*Type* Required [XlAxisType](#). The axis type.

XlAxisType can be one of these XlAxisType constants.

**xlCategory**

**xlSeriesAxis** Valid only for 3-D charts.

**xlValue**

*AxisGroup* Optional [XlAxisGroup](#). The axis.

XlAxisGroup can be one of these XlAxisGroup constants.

**xlPrimary** *default*

**xlSecondary**

 [Item method as it applies to the \*\*Comments\*\*, \*\*ODBCErrors\*\*, \*\*OLEDBErrors\*\* and \*\*Points\*\* objects.](#)

Returns a single object from a collection.

*expression*.**Item**(*Index*)

*expression* Required. An expression that returns one of the above objects.

*Index* Required **Long**. The index number for the object.

 [Item method as it applies to the \*\*Names\*\* object.](#)

Returns a single [Name](#) object from a **Names** collection.

*expression*.**Item**(*Index*, *IndexLocal*, *RefersTo*)

*expression* Required. An expression that returns a **Names** collection.

**Index** Optional **Variant**. The name or number of the defined name to be returned.

**IndexLocal** Optional **Variant**. The name of the defined name, in the language of the user. No names will be translated if you use this argument.

**RefersTo** Optional **Variant**. What the name refers to. You use this argument to identify a name by what it refers to.

## Remarks

You must specify one, and only one, of these three arguments.

[Item method as it applies to all other objects in the Applies To list.](#)

Returns a single object from a collection.

*expression*.**Item**(*Index*)

*expression* Required. An expression that returns all other objects in the Applies To list.

**Index** Required **Variant**. The name or index number for the object.

## Remarks

The text name of the object is the value of the [Name](#) and [Value](#) properties. For an [Online Analytical Processing \(OLAP\)](#) data source, the value is equal to the value of the [SourceName](#) property, and for other data sources, the value is equal to the value of the [Caption](#) property.

## Example

 [As it applies to the \*\*Axes\*\* object.](#)

This example sets the title text for the category axis on Chart1.

```
With Charts("chart1").Axes.Item(xlCategory)
    .HasTitle = True
    .AxisTitle.Caption = "1994"
End With
```

 [As it applies to the \*\*CalculatedFields\*\* object.](#)

This example sets the formula for calculated field one.

```
Worksheets(1).PivotTables(1).CalculatedFields.Item(1) _
    .Formula = "=Revenue - Cost"
```

 [As it applies to the \*\*CalculatedItems\*\* and \*\*PivotItemList\*\* objects.](#)

This example hides calculated item one.

```
Worksheets(1).PivotTables(1).PivotFields("year") _
    .CalculatedItems.Item(1).Visible = False
```

 [As it applies to the \*\*CanvasShapes\*\*, \*\*GroupShapes\*\*, and \*\*ShapeRange\*\* objects](#)

This example sets the **OnAction** property for shape two in a shape range. If the sr variable doesn't represent a **ShapeRange** object, this example fails.

```
Dim sr As Shape
sr.Item(2).OnAction = "ShapeAction"
```

 [As it applies to the \*\*ChartGroups\*\* object.](#)

This example adds drop lines to chart group one on chart sheet one.

```
Charts(1).ChartGroups.Item(1).HasDropLines = True
```

[As it applies to the \*\*ChartObjects\*\* object.](#)

This example activates embedded chart one.

```
Worksheets("sheet1").ChartObjects.Item(1).Activate
```

[As it applies to the \*\*Comments\*\* object.](#)

This example hides comment two.

```
Worksheets(1).Comments.Item(2).Visible = False
```

[As it applies to the \*\*CustomViews\*\* object.](#)

This example includes print settings in the custom view named Current Inventory.

```
ThisWorkbook.CustomViews.Item("Current Inventory") _  
    .PrintSettings = True
```

[As it applies to the \*\*DataLabels\*\* object.](#)

This example sets the number format for the fifth data label in series one in embedded chart one on worksheet one.

```
Worksheets(1).ChartObjects(1).Chart _  
    .SeriesCollection(1).DataLabels.Item(5).NumberFormat = "0.000"
```

[As it applies to the \*\*FormatConditions\*\* object.](#)

This example sets format properties for an existing conditional format for cells E1:E10.

```
With Worksheets(1).Range("e1:e10").FormatConditions.Item(1)
    With .Borders
        .LineStyle = xlContinuous
        .Weight = xlThin
        .ColorIndex = 6
    End With
End With
```

[As it applies to the \*\*LegendEntries\*\* object.](#)

This example changes the font for the text of the legend entry at the top of the legend (this is usually the legend for series one) in embedded chart one on Sheet1.

```
Worksheets("sheet1").ChartObjects(1).Chart _
    .Legend.LegendEntries.Item(1).Font.Italic = True
```

[As it applies to the \*\*Names\*\* object.](#)

This example deletes the name mySortRange from the active workbook.

```
ActiveWorkbook.Names.Item("mySortRange").Delete
```

[As it applies to the \*\*ODBCErrors\*\* object.](#)

This example displays an ODBC error.

```
Set er = Application.ODBCErrors.Item(1)
MsgBox "The following error occurred:" &
    er.ErrorString & " : " & er.SqlState
```

[As it applies to the \*\*OLEDBErrors\*\* object.](#)

This example displays an OLE DB error.

```
Set objEr = Application.OLEDBErrors.Item(1)
MsgBox "The following error occurred:" & _
    objEr.ErrorString & " : " & objEr.SqlState
```

[As it applies to the \*\*OLEObjects\*\* object.](#)

This example deletes OLE object one from Sheet1.

```
Worksheets("sheet1").OLEObjects.Item(1).Delete
```

[As it applies to the \*\*Parameters\*\* object.](#)

This example modifies the parameter prompt string.

```
With Worksheets(1).QueryTables(1).Parameters.Item(1)  
    .SetParam xlPrompt, "Please " & .PromptString  
End With
```

[As it applies to the \*\*PivotCaches\*\* object.](#)

This example refreshes cache one.

```
ActiveWorkbook.PivotCaches.Item(1).Refresh
```

[As it applies to the \*\*PivotFields\*\* object.](#)

This example makes the Year field a row field in the first PivotTable report on Sheet3.

```
Worksheets("sheet3").PivotTables(1) _  
    .PivotFields.Item("year").Orientation = xlRowField
```

[As it applies to the \*\*PivotFormulas\*\* object.](#)

This example displays the first formula for PivotTable one on worksheet one.

```
MsgBox Worksheets(1).PivotTables(1).PivotFormulas.Item(1).Formula
```

[As it applies to the \*\*PivotItems\*\* object.](#)

This example hides the 1998 item in the first PivotTable report on Sheet3.

```
Worksheets("sheet3").PivotTables(1) _  
    .PivotFields("year").PivotItems.Item("1998").Visible = False
```

[As it applies to the \*\*PivotTables\*\* object.](#)

This example makes the Year field a row field in the first PivotTable report on Sheet3.

```
Worksheets("sheet3").PivotTables.Item(1) _  
    .PivotFields("year").Orientation = xlRowField
```

[As it applies to the \*\*Points\*\* object.](#)

This example sets the marker style for the third point in series one in embedded chart one on worksheet one. The specified series must be a 2-D line, scatter, or radar series.

```
Worksheets(1).ChartObjects(1).Chart. _  
    SeriesCollection(1).Points.Item(3).MarkerStyle = xlDiamond
```

[As it applies to the \*\*QueryTables\*\* object.](#)

This example sets a query table so that formulas to the right of the query table are automatically updated whenever it's refreshed.

```
Sheets("sheet1").QueryTables.Item(1).FillAdjacentFormulas = True
```

[As it applies to the \*\*Scenarios\*\* object.](#)

This example shows the scenario named Typical on the worksheet named Options.

```
Worksheets("options").Scenarios.Item("typical").Show
```

[As it applies to the \*\*SeriesCollection\*\* object.](#)

This example sets the number of units that the trendline on Chart1 extends forward and backward. The example should be run on a 2-D column chart that contains a single series with a trendline.

```
With Charts("Chart1").SeriesCollection.Item(1).Trendlines.Item(1)
    .Forward = 5
    .Backward = .5
End With
```

[As it applies to the \*\*Shapes\*\* object.](#)

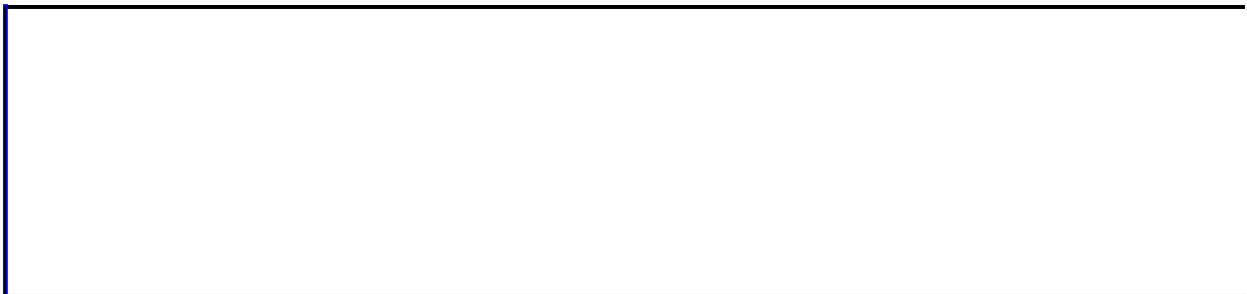
This example sets the **OnAction** property for shape two in a **Shapes** collection. If the ss variable doesn't represent a **Shapes** object, this example fails.

```
Dim ss As Shape
ss.Item(2).OnAction = "ShapeAction"
```

[As it applies to the \*\*Trendlines\*\* object.](#)

This example sets the number of units that the trendline on Chart1 extends forward and backward. The example should be run on a 2-D column chart that contains a single series with a trendline.

```
With Charts("Chart1").SeriesCollection(1).Trendlines.Item(1)
    .Forward = 5
    .Backward = .5
End With
```



# Justify Method

Rearranges the text in a range so that it fills the range evenly.

*expression*.**Justify**

*expression* Required. An expression that returns a **Range** object.

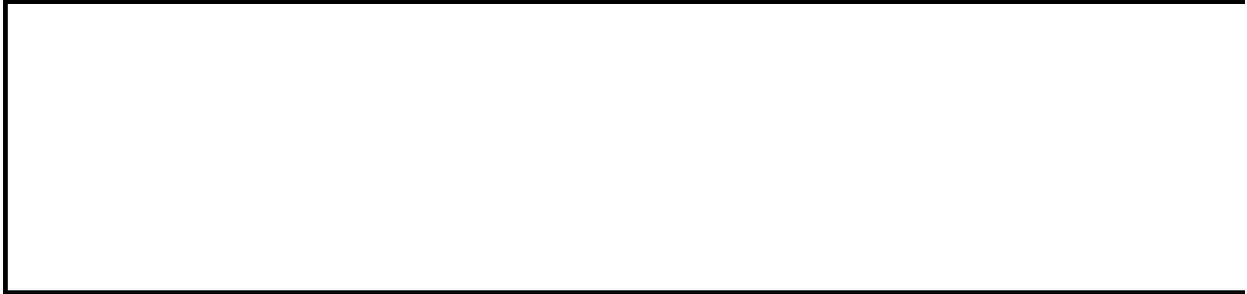
## Remarks

If the range isn't large enough, Microsoft Excel displays a message telling you that text will extend below the range. If you click the **OK** button, justified text will replace the contents in cells that extend beyond the selected range. To prevent this message from appearing, set the [DisplayAlerts](#) property to **False**. After you set this property, text will always replace the contents in cells below the range.

## Example

This example justifies the text in cell A1 on Sheet1.

```
Worksheets("Sheet1").Range("A1").Justify
```



# LargeScroll Method

Scrolls the contents of the window by pages.

*expression*.**LargeScroll**(*Down*, *Up*, *ToRight*, *ToLeft*)

*expression* Required. An expression that returns a **Window** object.

**Down** Optional **Variant**. The number of pages to scroll the contents down.

**Up** Optional **Variant**. The number of pages to scroll the contents up.

**ToRight** Optional **Variant**. The number of pages to scroll the contents to the right.

**ToLeft** Optional **Variant**. The number of pages to scroll the contents to the left.

## Remarks

If ***Down*** and ***Up*** are both specified, the contents of the window are scrolled by the difference of the arguments. For example, if ***Down*** is 3 and ***Up*** is 6, the contents are scrolled up three pages.

If ***ToLeft*** and ***ToRight*** are both specified, the contents of the window are scrolled by the difference of the arguments. For example, if ***ToLeft*** is 3 and ***ToRight*** is 6, the contents are scrolled to the right three pages.

Any of the arguments can be a negative number.

## Example

This example scrolls the contents of the active window of Sheet1 down three pages.

```
Worksheets("Sheet1").Activate  
Activewindow.LargeScroll down:=3
```



# LegendEntries Method

Returns an object that represents either a single legend entry (a [LegendEntry](#) object) or a collection of legend entries (a [LegendEntries](#) object) for the legend.

*expression*.**LegendEntries**(*Index*)

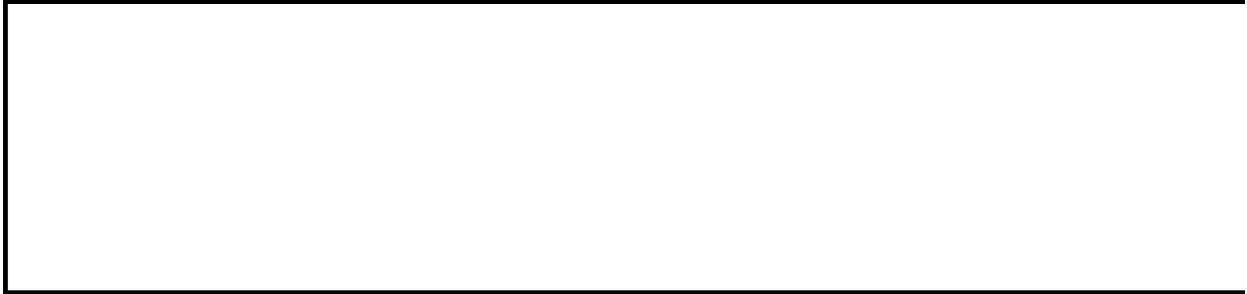
*expression* Required. An expression that returns a **Legend** object.

*Index* Optional **Variant**. The number of the legend entry.

## Example

This example sets the font for legend entry one on Chart1.

```
Charts("Chart1").Legend.LegendEntries(1).Font.Name = "Arial"
```



# LineGroups Method

On a 2-D chart, returns an object that represents either a single line chart group (a [ChartGroup](#) object) or a collection of the line chart groups (a [ChartGroups](#) collection).

*expression*.**LineGroups**(*Index*)

*expression* Required. An expression that returns a **Chart** object.

*Index* Optional **Variant**. Specifies the chart group.

## Example

This example sets line group one in Chart1 to use a different color for each data marker. The example should be run on a 2-D chart.

```
Charts("Chart1").LineGroups(1).VaryByCategories = True
```



[Show All](#)

# LinkInfo Method

Returns the link date and update status. **Variant**.

*expression*.**LinkInfo**(*Name*, *LinkInfo*, *Type*, *EditionRef*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*Name* Optional **String**.

*LinkInfo* Required [XLLinkInfo](#). The type of information to be returned.

XLLinkInfo can be one of these XLLinkInfo constants.

**xlEditionDate**

**xlLinkInfoStatus**

**xlUpdateState** This method returns 1 if the link updates automatically, or it returns 2 if the link must be updated manually.

*Type* Optional [XLLinkInfoType](#). The type of link to return.

XLLinkInfoType can be one of these XLLinkInfoType constants.

**xlLinkInfoOLELinks** (also handles DDE links)

**xlLinkInfoPublishers**

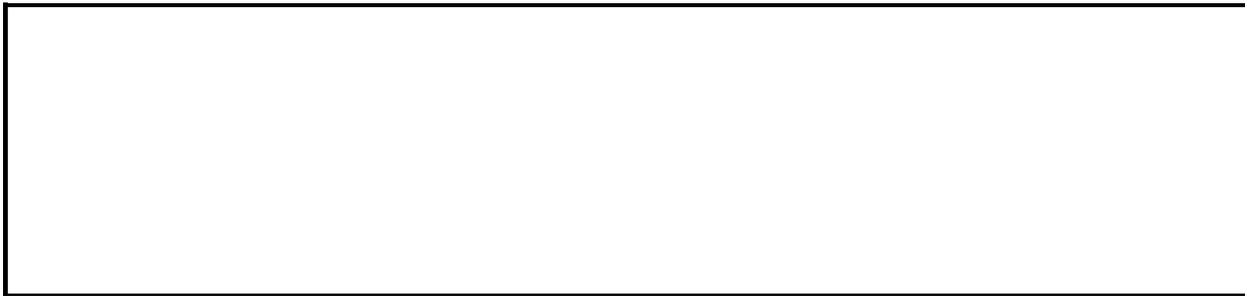
**xlLinkInfoSubscribers**

*EditionRef* Optional **Variant**. If the link is an edition, this argument specifies the edition reference as a string in R1C1 style. This argument is required if there's more than one publisher or subscriber with the same name in the workbook.

## Example

This example displays a message box if the link is updated automatically.

```
If ActiveWorkbook.LinkInfo( _  
    "Word.Document|Document1!'!DDE_LINK1", xlUpdateState, _  
    xlOLELinks) = 1 Then  
    MsgBox "Link updates automatically"  
End If
```



[Show All](#)

# LinkSources Method

Returns an array of links in the workbook. The names in the array are the names of the linked documents, editions, or DDE or OLE servers. Returns **Empty** if there are no links. **Variant**.

*expression*.**LinkSources**(*Type*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Type** Optional [XLLink](#). The type of link to return.

XLLink can be one of these XLLink constants.

**xlExcelLinks**

**xlOLELinks** (also handles DDE links)

**xlPublishers**

**xlSubscribers**

## Remarks

The format of the array is a one-dimensional array for all types but publisher and subscriber. The returned strings contain the name of the link source, in the appropriate notation for the link type. For example, DDE links use the "Server|Document!Item" syntax.

For publisher and subscriber links, the returned array is two-dimensional. The first column of the array contains the names of the edition, and the second column contains the references of the editions as text.

## Example

This example displays a list of OLE and DDE links in the active workbook. The example should be run on a workbook that contains one or more linked Word objects.

```
aLinks = ActiveWorkbook.LinkSources(xlOLELinks)
If Not IsEmpty(aLinks) Then
    For i = 1 To UBound(aLinks)
        MsgBox "Link " & i & ":" & Chr(13) & aLinks(i)
    Next i
End If
```



# List Method

Returns or sets the text entries in the specified list box or a combo box, as an array of strings, or returns or sets a single text entry. An error occurs if there are no entries in the list.

*expression*.**List**(*Index*)

*expression* Required. An expression that returns a **ControlFormat** object.

*Index* Optional **Variant**. The index number of a single text entry to be set or returned. If this argument is omitted, the entire list is returned or set as an array of strings.

## Remarks

Setting this property clears any range specified by the **ListFillRange** property.

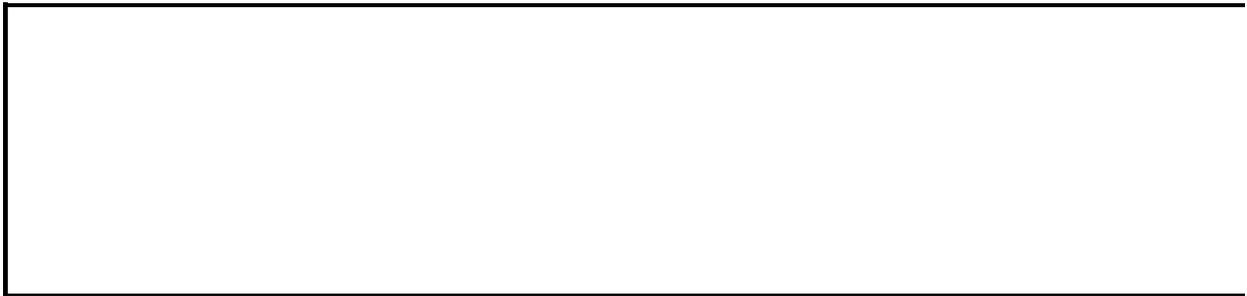
## Example

This example sets the entries in a list box on worksheet one. If Shapes(2) doesn't represent a list box, this example fails.

```
Worksheets(1).Shapes(2).ControlFormat.List = _  
    Array("cogs", "widgets", "sprockets", "gizmos")
```

This example sets entry four in a list box on worksheet one. If Shapes(2) doesn't represent a list box, this example fails.

```
Worksheets(1).Shapes(2).ControlFormat.List(4) = "gadgets"
```



[Show All](#)

# ListFormulas Method

Creates a list of calculated PivotTable items and fields on a separate worksheet.

*expression*.**ListFormulas**

*expression* Required. An expression that returns a **PivotTable** object.

## Remarks

This method isn't available for [OLAP](#) data sources.

## Example

This example creates a list of calculated items and fields for the first PivotTable report on worksheet one.

```
Worksheets(1).PivotTables(1).ListFormulas
```



# ListNames Method

Pastes a list of all nonhidden names onto the worksheet, beginning with the first cell in the range.

*expression*.**ListNames**

*expression* Required. An expression that returns a **Worksheet** object.

## Remarks

Use the [Names](#) property to return a collection of all the names on a worksheet.

## Example

This example pastes a list of defined names into cell A1 on Sheet1. The example pastes both workbook-level names and sheet-level names defined on Sheet1.

```
Worksheets("Sheet1").Range("A1").ListNames
```



[Show All](#)

# LoadSettings Method

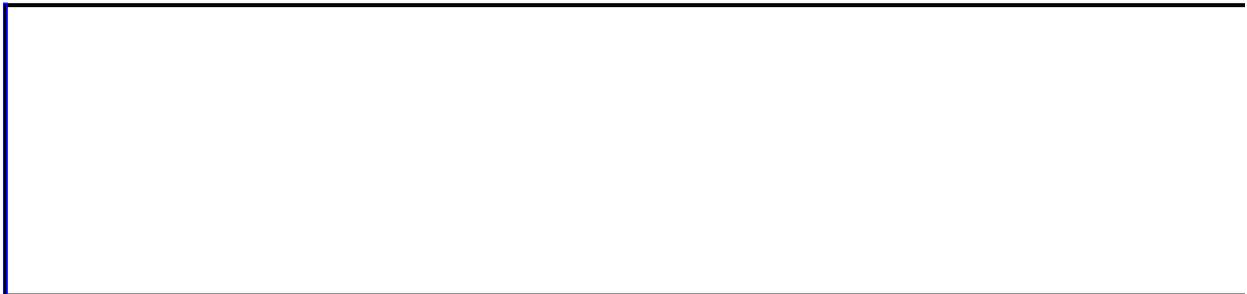
**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Initializes the specified data binding with settings from an XML data file or a Data Retrieval Service Connection (.uxdc) file.

*expression*.**LoadSettings**(*Url*)

*expression* Required. An expression that returns an [XmlDataBinding](#) object.

**Url** Required **String**. The path to the XML data or Microsoft Office XML Data Source (.uxdc) file. The path is specified in the [Uniform Resource Locator \(URL\)](#) or [universal naming convention \(UNC\)](#) format.



[Show All](#)

# Location Method

Moves the chart to a new location. **Chart** object.

*expression*.**Location**(*Where*, *Name*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*Where* Required [XLChartLocation](#). Where to move the chart.

XLChartLocation can be one of these XLChartLocation constants.

**xLLocationAsNewSheet**

**xLLocationAsObject**

**xLLocationAutomatic**

*Name* Optional **Variant**; required if *Where* is **xLLocationAsObject**. The name of the sheet where the chart will be embedded if *Where* is **xLLocationAsObject** or the name of the new sheet if *Where* is **xLLocationAsNewSheet**.

## Example

This example moves the embedded chart to a new chart sheet named "Monthly Sales."

```
Worksheets(1).ChartObjects(1).Chart _  
    .Location xlLocationAsNewSheet, "Monthly Sales"
```



# MacroOptions Method

Corresponds to options in the **Macro Options** dialog box. You can also use this method to display a user defined function (UDF) in a built-in or new category within the **Insert Function** dialog box.

*expression*.**MacroOptions**(*Macro*, *Description*, *HasMenu*, *MenuText*, *HasShortcutKey*, *ShortcutKey*, *Category*, *StatusBar*, *HelpContextID*, *HelpFile*)

*expression* Required. An expression that returns an **Application** object.

**Macro** Optional **Variant**. The macro name or the name of a user defined function (UDF).

**Description** Optional **Variant**. The macro description.

**HasMenu** Optional **Variant**. This argument is ignored.

**MenuText** Optional **Variant**. This argument is ignored.

**HasShortcutKey** Optional **Variant**. **True** to assign a shortcut key to the macro (**ShortcutKey** must also be specified). If this argument is **False**, no shortcut key is assigned to the macro. If the macro already has a shortcut key, setting this argument to **False** removes the shortcut key. The default value is **False**.

**ShortcutKey** Optional **Variant**. Required if **HasShortcutKey** is **True**; ignored otherwise. The shortcut key.

**Category** Optional **Variant**. An integer that specifies an existing macro function category (Financial, Date & Time, or User Defined, for example). See the remarks section to determine the integers that are mapped to the built-in categories. You can also specify a string for a custom category. If you provide a string it will be treated as the category name that is displayed in the **Insert Function** dialog box. If the category name has never been used, a new category is defined with that name. If you use a category name that is that same a built-in name, Excel will map the user defined function to that built-in category.

**StatusBar** Optional **Variant**. The status bar text for the macro.

**HelpContextId** Optional **Variant**. An integer that specifies the context ID for the Help topic assigned to the macro.

**HelpFile** Optional **Variant**. The name of the Help file that contains the Help topic defined by **HelpContextId**.

## Remarks

The following table lists which integers are mapped to the built-in categories that can be used in the *Category* parameter.

### Integer Category

1	<b>Financial</b>
2	<b>Date &amp; Time</b>
3	<b>Math &amp; Trig</b>
4	<b>Statistical</b>
5	<b>Lookup &amp; Reference</b>
6	<b>Database</b>
7	<b>Text</b>
8	<b>Logical</b>
9	<b>Information</b>
10	<b>Commands</b>
11	<b>Customizing</b>
12	<b>Macro Control</b>
13	<b>DDE/External</b>
14	<b>User Defined</b>
15	First custom category
16	Second custom category
17	Third custom category
18	Fourth custom category
19	Fifth custom category
20	Sixth custom category
21	Seventh custom category
22	Eighth custom category
23	Ninth custom category
24	Tenth custom category
25	Eleventh custom category
26	Twelfth custom category

- 27 Thirteenth custom category
- 28 Fourteenth custom category
- 29 Fifteenth custom category
- 30 Sixteenth custom category
- 31 Seventeenth custom category
- 32 Eighteenth custom category

## Example

This example adds a user-defined macro called "TestMacro" to a custom category named "My Custom Category". After you run this example, you should see "My Custom Category" which contains the "TestMacro" user-defined function in the **Or select a category** drop-down list in the **Insert Function** dialog box.

```
Function TestMacro()  
    MsgBox ActiveWorkbook.Name  
End Function
```

```
Sub AddUDFToCustomCategory()  
    Application.MacroOptions Macro:="TestMacro", Category:="My Custom Category"  
End Sub
```



# MailLogoff Method

Closes a MAPI mail session established by Microsoft Excel.

*expression*.**MailLogoff**

*expression* Required. An expression that returns an **Application** object.

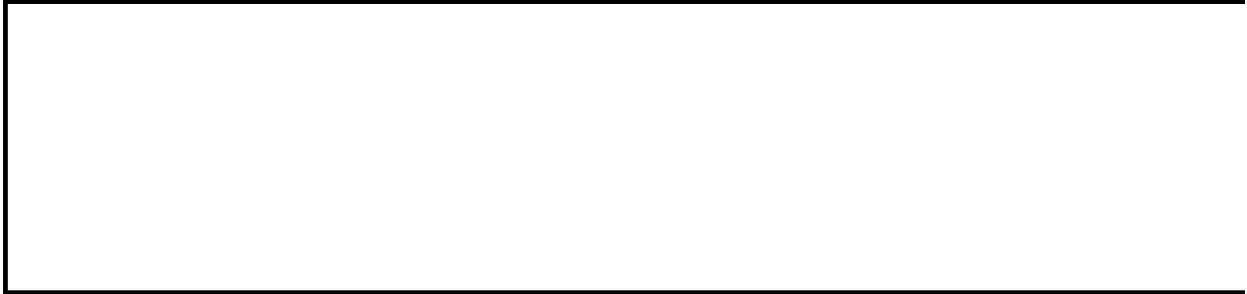
## **Remarks**

You cannot use this method to close or log off Microsoft Mail.

## Example

This example closes the established mail session, if there is one.

```
If Not IsNull(Application.MailSession) Then Application.MailLogoff
```



# MailLogon Method

Logs in to MAPI Mail or Microsoft Exchange and establishes a mail session. If Microsoft Mail isn't already running, you must use this method to establish a mail session before mail or document routing functions can be used.

*expression*.**MailLogon**(*Name*, *Password*, *DownloadNewMail*)

*expression* Required. An expression that returns an **Application** object.

**Name** Optional **VARIANT**. The mail account name or Microsoft Exchange profile name. If this argument is omitted, the default mail account name is used.

**Password** Optional **VARIANT**. The mail account password. This argument is ignored in Microsoft Exchange.

**DownloadNewMail** Optional **VARIANT**. **True** to download new mail immediately.

## **Remarks**

Microsoft Excel logs off any mail sessions it previously established before attempting to establish the new session.

To piggyback on the system default mail session, omit both the name and password parameters.

## Example

This example logs in to the default mail account.

```
If IsNull(Application.MailSession) Then  
    Application.MailLogon  
End If
```



# MakeConnection Method

Establishes a connection for the specified PivotTable cache.

*expression*.**MakeConnection**

*expression* Required. An expression that returns a **PivotCache** object.

## Remarks

The **MakeConnection** method can be used after the cache drops a connection and the user wants to re-establish the connection.

Various objects and methods might return a run-time error if the cache is not connected. Use of this method assures a connection before executing other objects or methods.

This method will result in a run-time error if the **MaintainConnection** property of the specified PivotTable cache has been set to False, the **SourceType** property of the specified PivotTable cache has not been set to xlExternal, or if the connection is not OLEDB.

**Note** Microsoft Excel might drop a connection temporarily in the course of a session (unknown to the VBA programmer), so this method proves useful.

## Example

The following example determines if the cache is connected to its source and makes a connection to the source if necessary. This example assumes a PivotTable cache exists on the active worksheet.

```
Sub UseMakeConnection()  
  
    Dim pvtCache As PivotCache  
  
    Set pvtCache = Application.ActiveWorkbook.PivotCaches.Item(1)  
  
    ' Handle run-time error if external source is not an OLEDB data  
    On Error GoTo Not_OLEDB  
  
    ' Check connection setting and make connection if necessary.  
    If pvtCache.IsConnected = True Then  
        MsgBox "The MakeConnection method is not needed."  
    Else  
        pvtCache.MakeConnection  
        MsgBox "A connection has been made."  
    End If  
    Exit Sub  
  
Not_OLEDB:  
    MsgBox "The data source is not an OLEDB data source"  
  
End Sub
```



[Show All](#)

# Merge Method

 [Merge method as it applies to the \*\*Scenarios\*\* object.](#)

Merges the scenarios from another sheet into the **Scenarios** collection.

*expression*.**Merge**(*Source*)

*expression* Required. An expression that returns the **Scenarios** object.

**Source** Required **Variant**. The name of the sheet that contains scenarios to be merged, or a **Worksheet** object that represents that sheet.

 [Merge method as it applies to the \*\*Styles\*\* object.](#)

Merges the styles from another workbook into the **Styles** collection.

*expression*.**Merge**(*Workbook*)

*expression* Required. An expression that returns the **Styles** object.

**Workbook** Required **Variant**. A **Workbook** object that represents the workbook containing styles to be merged.

 [Merge method as it applies to the \*\*Range\*\* object.](#)

Creates a merged cell from the specified **Range** object.

*expression*.**Merge**(*Across*)

*expression* Required. An expression that returns the **Range** object.

**Across** Optional **Variant**. **True** to merge cells in each row of the specified range as separate merged cells. The default value is **False**.

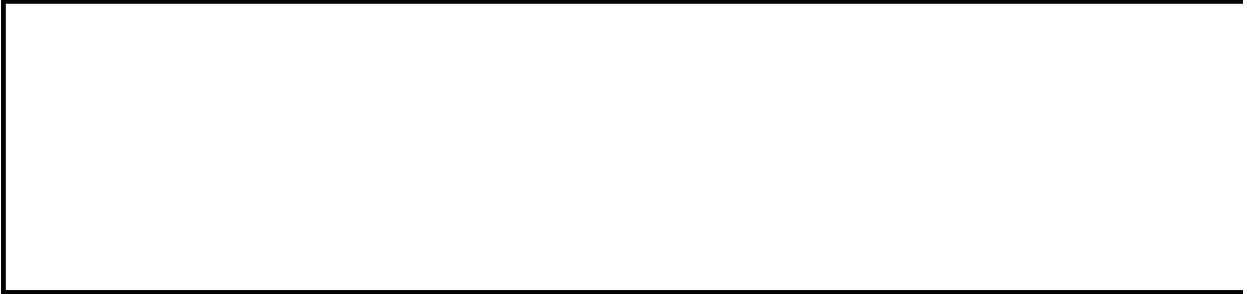
## Remarks

The value of a merged range is specified in the cell of the range's upper-left corner.

## Example

This example merges the styles from the workbook Template.xls into the active workbook.

```
ActiveWorkbook.Styles.Merge Workbook:=Workbooks("TEMPLATE.XLS")
```



# MergeWorkbook Method

Merges changes from one workbook into an open workbook.

*expression*.MergeWorkbook(*Filename*)

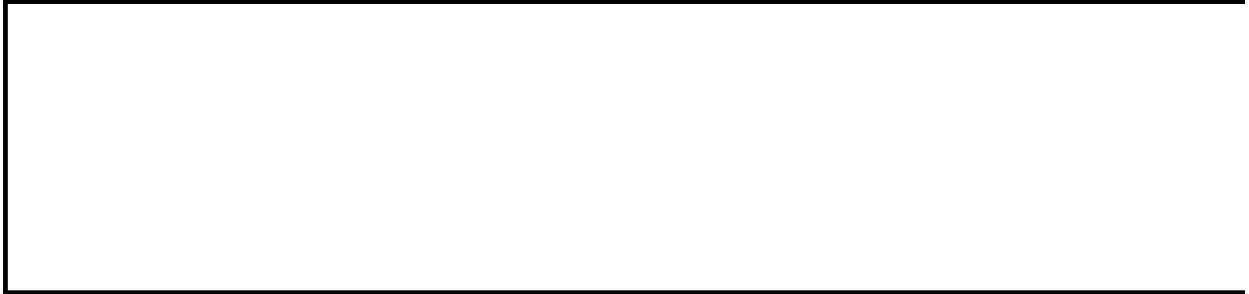
*expression* Required. An expression that returns a **Workbook** object.

**Filename** Required **String**. The file name of the workbook that contains the changes to be merged into the open workbook.

## Example

This example merges changes from Book1.xls into the active workbook.

```
ActiveWorkbook.MergeWorkbook "Book1.xls"
```



# Modify Method

Modifies the data validation or conditional format. For more information, click the object you want to modify.

[FormatCondition](#)

[Validation](#)

---

---

# Move Method

Moves the sheet to another location in the workbook.

*expression*.**Move**(*Before*, *After*)

*expression* Required. An expression that returns an object in the Applies To list.

**Before** Optional **Variant**. The sheet before which the moved sheet will be placed. You cannot specify **Before** if you specify **After**.

**After** Optional **Variant**. The sheet after which the moved sheet will be placed. You cannot specify **After** if you specify **Before**.

## Remarks

If you don't specify either ***Before*** or ***After***, Microsoft Excel creates a new workbook that contains the moved sheet.

## Example

This example moves Sheet1 after Sheet3 in the active workbook.

```
Worksheets("Sheet1").Move _  
    after:=Worksheets("Sheet3")
```



[Show All](#)

# MoveNode Method

Moves a diagram node and any of its child nodes, within a diagram.

*expression*.**MoveNode**(*pTargetNode*, *pos*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*pTargetNode* Required **DiagramNode** object. The diagram node where the specified node will be moved.

*pos* Required [MsoRelativeNodePosition](#). The position to move the node, relative to *TargetNode*.

MsoRelativeNodePosition can be one of these MsoRelativeNodePosition constants.

**msoAfterLastSibling**

**msoAfterNode**

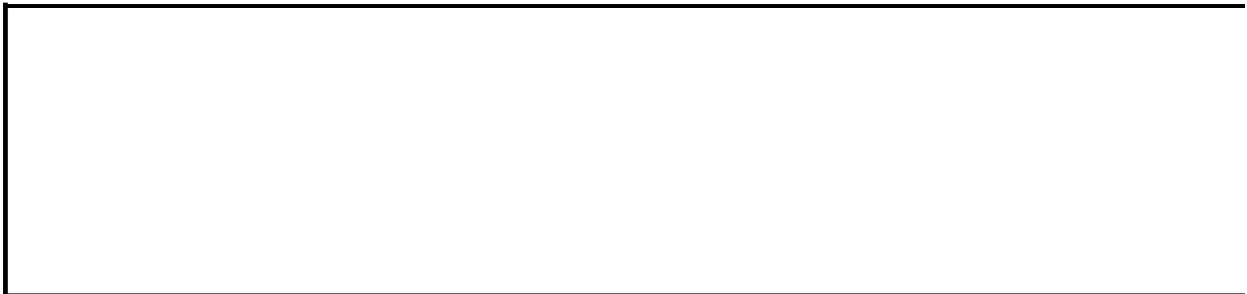
**msoBeforeFirstSibling**

**msoBeforeNode**

## Example

The following example moves the second diagram node of a newly-created diagram to the last node.

```
Sub MoveDiagramNode()  
  
    Dim dgnNode As DiagramNode  
    Dim shpDiagram As Shape  
    Dim intCount As Integer  
  
    'Add pyramid diagram to the current document  
    Set shpDiagram = ActiveSheet.Shapes.AddDiagram( _  
        Type:=msoDiagramPyramid, Left:=10, _  
        Top:=15, Width:=400, Height:=475)  
  
    'Add four child nodes to the pyramid diagram  
    Set dgnNode = shpDiagram.DiagramNode.Children.AddNode  
  
    For intCount = 1 To 3  
        dgnNode.AddNode  
    Next intCount  
  
    'Move the second node to after where the  
    'fourth node is currently located.  
    dgnNode.Diagram.Nodes(2).MoveNode _  
        pTargetNode:=dgnNode.Diagram.Nodes(4), _  
        Pos:=msoAfterLastSibling  
  
End Sub
```



# NavigateArrow Method

Navigates a tracer arrow for the specified range to the precedent, dependent, or error-causing cell or cells. Selects the precedent, dependent, or error cells and returns a **Range** object that represents the new selection. This method causes an error if it's applied to a cell without visible tracer arrows.

*expression.NavigateArrow(TowardPrecedent, ArrowNumber, LinkNumber)*

*expression* Required. An expression that returns a **Range** object.

**TowardPrecedent** Optional **Variant**. Specifies the direction to navigate: **True** to navigate toward precedents, **False** to navigate toward dependent.

**ArrowNumber** Optional **Variant**. Specifies the arrow number to navigate; corresponds to the numbered reference in the cell's formula.

**LinkNumber** Optional **Variant**. If the arrow is an external reference arrow, this argument indicates which external reference to follow. If this argument is omitted, the first external reference is followed.

## Example

This example navigates along the first tracer arrow from cell A1 on Sheet1 toward the precedent cell. The example should be run on a worksheet containing a formula in cell A1 that includes references to cells D1, D2, and D3 (for example, the formula =D1\*D2\*D3). Before running the example, display the **Auditing** toolbar, select cell A1, and click the **Trace Precedents** button.

```
Worksheets("Sheet1").Activate  
Range("A1").Select  
ActiveCell.NavigateArrow True, 1
```



# NewSeries Method

Creates a new series. Returns a **Series** object that represents the new series.

*expression*.NewSeries()

*expression* Required. An expression that returns a **SeriesCollection** object.

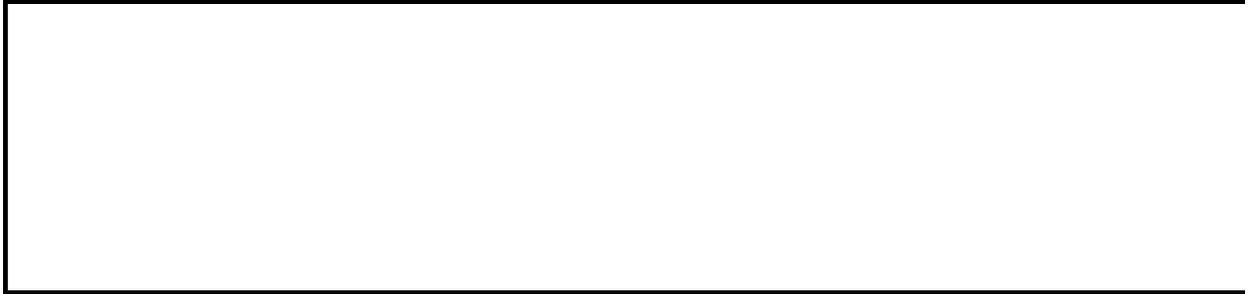
## Remarks

This method isn't available for PivotChart reports.

## Example

This example adds a new series to chart one.

```
Set ns = Charts(1).SeriesCollection.NewSeries
```



# NewWindow Method

Creates a new window or a copy of the specified window.

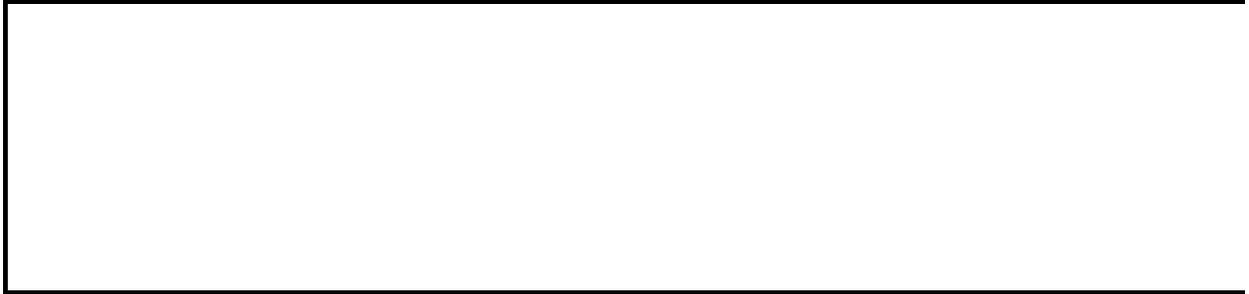
*expression*.**NewWindow**

*expression* Required. An expression that returns a **Window** or **Workbook** object.

## Example

This example creates a new window for the active workbook.

`ActiveWorkbook.NewWindow`



# Next Method

Returns a **Comment** object that represents the next comment.

*expression*.**Next**

*expression* Required. An expression that returns a **Comment** object.

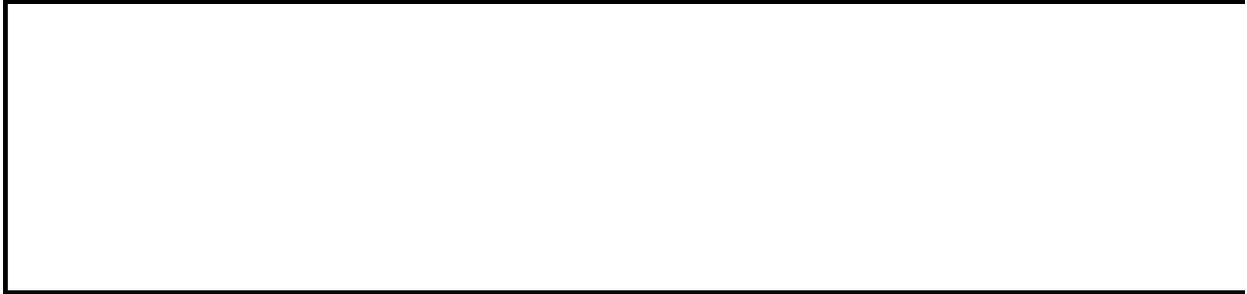
## Remarks

This method works only on one sheet. Using this method on the last comment on a sheet returns **Null** (not the next comment on the next sheet).

## Example

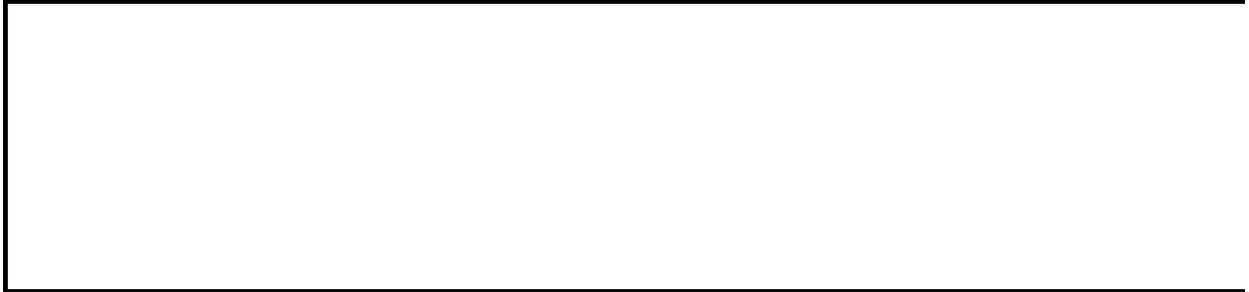
This example hides the next comment.

```
Range("a1").Comment.Next.Visible = False
```



# NextLetter Method

You have requested Help for a Visual Basic keyword used only on the Macintosh. For information about this keyword, consult the language reference Help included with Microsoft Office Macintosh Edition.



# NextNode Method

Selects the next diagram node in a series of nodes. Returns a **DiagramNode** object representing the newly-selected node.

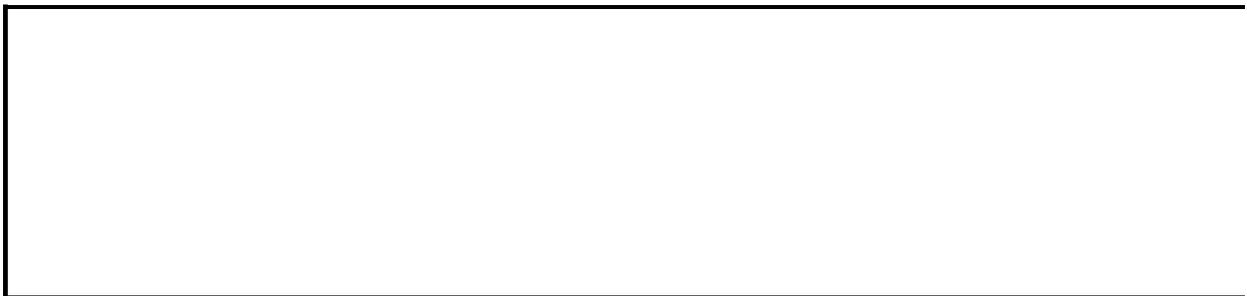
*expression*.**NextNode**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example creates an organization chart, and adds child nodes to the middle diagram node.

```
Sub AddChildrenToMiddle()  
  
    Dim dgnRoot As DiagramNode  
    Dim shpDiagram As Shape  
    Dim dgnNext As DiagramNode  
    Dim intCount As Integer  
  
    'Add organization chart to current document  
    Set shpDiagram = ActiveSheet.Shapes.AddDiagram( _  
        Type:=msoDiagramOrgChart, Left:=10, _  
        Top:=15, Width:=400, Height:=475)  
  
    'Add three child nodes to organization chart  
    Set dgnRoot = shpDiagram.DiagramNode.Children.AddNode  
  
    For intCount = 1 To 3  
        dgnRoot.Children.AddNode  
    Next  
  
    'Access the node immediately following  
    'the first diagram node  
    Set dgnNext = dgnRoot.Children.Item(1).NextNode  
  
    'Add three child nodes to the node immediately  
    'following the first diagram node  
    For intCount = 1 To 3  
        dgnNext.Children.AddNode  
    Next intCount  
  
End Sub
```



# NoteText Method

Returns or sets the cell note associated with the cell in the upper-left corner of the range. Read/write **String**.

Cell notes have been replaced by range comments. For more information, see the [Comment](#) object.

*expression*.**NoteText**(*Text*, *Start*, *Length*)

*expression* Required. An expression that returns a **Range** object.

**Text** Optional **Variant**. The text to add to the note (up to 255 characters). The text is inserted starting at position **Start**, replacing **Length** characters of the existing note. If this argument is omitted, this method returns the current text of the note starting at position **Start**, for **Length** characters.

**Start** Optional **Variant**. The starting position for the text that's set or returned. If this argument is omitted, this method starts at the first character. To append text to the note, specify a number larger than the number of characters in the existing note.

**Length** Optional **Variant**. The number of characters to be set or returned. If this argument is omitted, Microsoft Excel sets or returns characters from the starting position to the end of the note (up to 255 characters). If there are more than 255 characters from **Start** to the end of the note, this method returns only 255 characters.

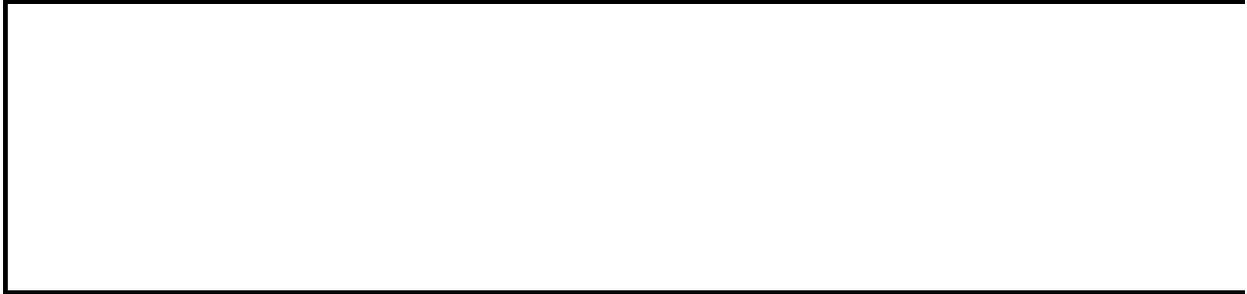
## **Remarks**

To add a note that contains more than 255 characters, use this method once to specify the first 255 characters, and then use it again to append the remainder of the note (no more than 255 characters at a time).

## Example

This example sets the cell note text for cell A1 on Sheet1.

```
Worksheets("Sheet1").Range("A1").NoteText "This may change!"
```



# OLEObjects Method

Returns an object that represents either a single OLE object (an [OLEObject](#) ) or a collection of all OLE objects (an [OLEObjects](#) collection) on the chart or sheet. Read-only.

*expression*.OLEObjects(*Index*)

*expression* Required. An expression that returns a **Chart** or **Worksheet** object.

*Index* Optional **Variant**. The name or number of the OLE object.

## Example

This example creates a list of link types for OLE objects on Sheet1. The list appears on a new worksheet created by the example.

```
Set newSheet = Worksheets.Add
i = 2
newSheet.Range("A1").Value = "Name"
newSheet.Range("B1").Value = "Link Type"
For Each obj In Worksheets("Sheet1").OLEObjects
    newSheet.Cells(i, 1).Value = obj.Name
    If obj.OLEType = xlOLELink Then
        newSheet.Cells(i, 2) = "Linked"
    Else
        newSheet.Cells(i, 2) = "Embedded"
    End If
    i = i + 1
Next
```



[Show All](#)

# OneColorGradient Method

 [OneColorGradient method as it applies to the \*\*FillFormat\*\* object.](#)

Sets the specified fill to a one-color gradient.

*expression*.**OneColorGradient**(*Style*, *Variant*, *Degree*)

*expression* Required. An expression that returns one of the above objects.

*Style* Required [MsoGradientStyle](#).

MsoGradientStyle can be one of these MsoGradientStyle constants.

**msoGradientDiagonalDown**

**msoGradientDiagonalUp**

**msoGradientFromCenter**

**msoGradientFromCorner**

**msoGradientFromTitle**

**msoGradientHorizontal**

**msoGradientMixed**

**msoGradientVertical**

*Variant* Required **Integer**. The gradient variant. Can be a value from 1 through 4, corresponding to one of the four variants on the **Gradient** tab in the **Fill Effects** dialog box. If *GradientStyle* is **msoGradientFromCenter**, the *Variant* argument can only be 1 or 2.

*Degree* Required **Single**. The gradient degree. Can be a value from 0.0 (dark) through 1.0 (light).

 [OneColorGradient method as it applies to the \*\*ChartFillFormat\*\* object.](#)

Sets the specified fill to a one-color gradient.

*expression*.**OneColorGradient**(*Style*, *Variant*, *Degree*)

*expression* Required. An expression that returns one of the above objects.

*Style* Required [MsoGradientStyle](#).

MsoGradientStyle can be one of these MsoGradientStyle constants.

**msoGradientDiagonalDown**

**msoGradientDiagonalUp**

**msoGradientFromCenter**

**msoGradientFromCorner**

**msoGradientFromTitle**

**msoGradientHorizontal**

**msoGradientMixed**

**msoGradientVertical**

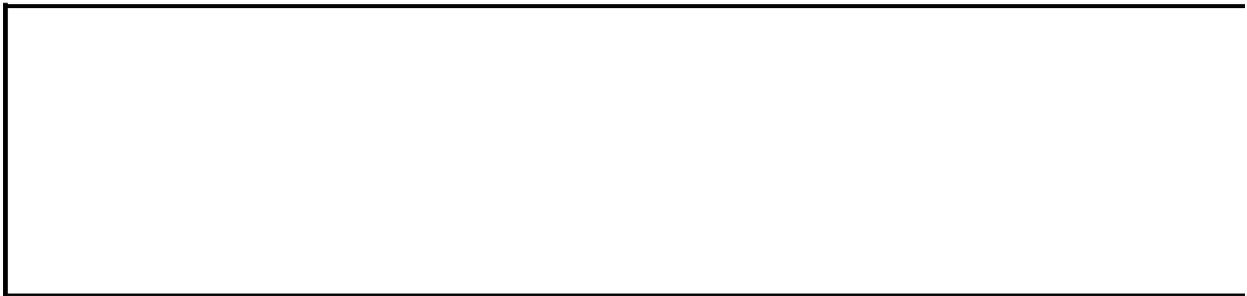
*Variant* Required **Long**. The gradient variant. Can be a value from 1 through 4, corresponding to one of the four variants on the **Gradient** tab in the **Fill Effects** dialog box. If *GradientStyle* is **msoGradientFromCenter**, the *Variant* argument can only be 1 or 2.

*Degree* Required **Single**. The gradient degree. Can be a value from 0.0 (dark) through 1.0 (light).

## Example

This example sets the fill format for chart two to the same style used for chart one.

```
Set c1f = Charts(1).ChartArea.Fill
If c1f.Type = msoFillGradient And _
    c1f.GradientColorType = msoGradientOneColor Then
    With Charts(2).ChartArea.Fill
        .Visible = True
        .OneColorGradient c1f.GradientStyle, _
            c1f.GradientVariant, c1f.GradientDegree
    End With
End If
```



# OnKey Method

Runs a specified procedure when a particular key or key combination is pressed.

*expression*.**OnKey**(*Key*, *Procedure*)

*expression* Required. An expression that returns an **Application** object.

**Key** Required **String**. A string indicating the key to be pressed.

**Procedure** Optional **Variant**. A string indicating the name of the procedure to be run. If **Procedure** is "" (empty text), nothing happens when **Key** is pressed. This form of **OnKey** changes the normal result of keystrokes in Microsoft Excel. If **Procedure** is omitted, **Key** reverts to its normal result in Microsoft Excel, and any special key assignments made with previous **OnKey** methods are cleared.

## Remarks

The **Key** argument can specify any single key combined with ALT, CTRL, or SHIFT, or any combination of these keys. Each key is represented by one or more characters, such as "a" for the character a, or "{ENTER}" for the ENTER key.

To specify characters that aren't displayed when you press the corresponding key (ENTER or TAB, for example), use the codes listed in the following table. Each code in the table represents one key on the keyboard.

Key	Code
BACKSPACE	{BACKSPACE} or {BS}
BREAK	{BREAK}
CAPS LOCK	{CAPSLOCK}
CLEAR	{CLEAR}
DELETE or DEL	{DELETE} or {DEL}
DOWN ARROW	{DOWN}
END	{END}
ENTER	~ (tilde)
ENTER (numeric keypad)	{ENTER}
ESC	{ESCAPE} or {ESC}
F1 through F15	{F1} through {F15}
HELP	{HELP}
HOME	{HOME}
INS	{INSERT}
LEFT ARROW	{LEFT}
NUM LOCK	{NUMLOCK}
PAGE DOWN	{PGDN}
PAGE UP	{PGUP}
RETURN	{RETURN}
RIGHT ARROW	{RIGHT}
SCROLL LOCK	{SCROLLLOCK}

TAB	{TAB}
UP ARROW	{UP}

You can also specify keys combined with SHIFT and/or CTRL and/or ALT. To specify a key combined with another key or keys, use the following table.

**To combine keys with Precede the key code by**

SHIFT	+ (plus sign)
CTRL	^ (caret)
ALT	% (percent sign)

To assign a procedure to one of the special characters (+, ^, %, and so on), enclose the character in braces. For details, see the example.

## Example

This example assigns "InsertProc" to the key sequence CTRL+PLUS SIGN and assigns "SpecialPrintProc" to the key sequence SHIFT+CTRL+RIGHT ARROW.

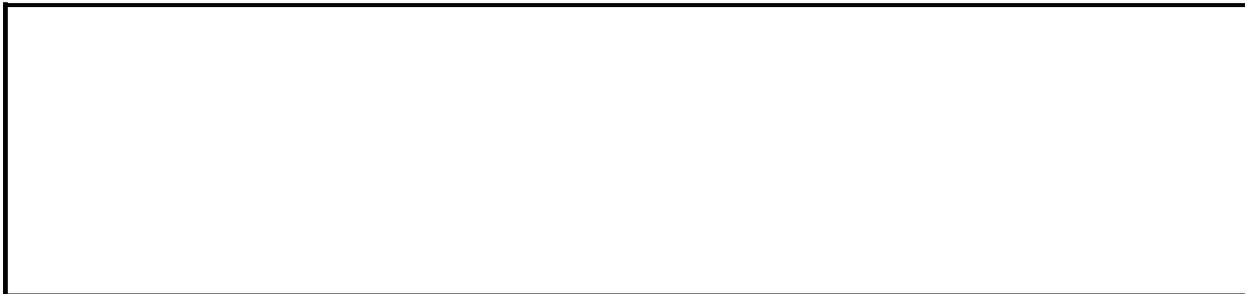
```
Application.OnKey "^{+}", "InsertProc"  
Application.OnKey "+^{RIGHT}", "SpecialPrintProc"
```

This example returns SHIFT+CTRL+RIGHT ARROW to its normal meaning.

```
Application.OnKey "+^{RIGHT}"
```

This example disables the SHIFT+CTRL+RIGHT ARROW key sequence.

```
Application.OnKey "+^{RIGHT}", ""
```



# OnRepeat Method

Sets the **Repeat** menu item and the name of the procedure that will run if you choose the **Repeat** command (**Edit** menu) after running the procedure that sets this property.

*expression*.**OnRepeat**(*Text*, *Procedure*)

*expression* Required. An expression that returns an **Application** object.

**Text** Required **String**. The text that appears with the **Repeat** command (**Edit** menu).

**Procedure** Required **String**. The name of the procedure that will be run when you choose the **Repeat** command (**Edit** menu).

## Remarks

If a procedure doesn't use the **OnRepeat** method, the **Repeat** command repeats procedure that was run most recently.

The procedure must use the **OnRepeat** and **OnUndo** methods last, to prevent the repeat and undo procedures from being overwritten by subsequent actions in the procedure.

## Example

This example sets the repeat and undo procedures.

```
Application.OnRepeat "Repeat VB Procedure", _  
    "Book1.xls!My_Repeat_Sub"  
Application.OnUndo "Undo VB Procedure", _  
    "Book1.xls!My_Undo_Sub"
```



# OnTime Method

Schedules a procedure to be run at a specified time in the future (either at a specific time of day or after a specific amount of time has passed).

*expression*.**OnTime**(*EarliestTime*, *Procedure*, *LatestTime*, *Schedule*)

*expression* Required. An expression that returns an **Application** object.

**EarliestTime** Required **Variant**. The time when you want this procedure to be run.

**Procedure** Required **String**. The name of the procedure to be run.

**LatestTime** Optional **Variant**. The latest time at which the procedure can be run. For example, if **LatestTime** is set to **EarliestTime** + 30 and Microsoft Excel is not in Ready, Copy, Cut, or Find mode at **EarliestTime** because another procedure is running, Microsoft Excel will wait 30 seconds for the first procedure to complete. If Microsoft Excel is not in Ready mode within 30 seconds, the procedure won't be run. If this argument is omitted, Microsoft Excel will wait until the procedure can be run.

**Schedule** Optional **Variant**. **True** to schedule a new **OnTime** procedure. **False** to clear a previously set procedure. The default value is **True**.

## Remarks

Use `Now + TimeValue(time)` to schedule something to be run when a specific amount of time (counting from now) has elapsed. Use `TimeValue(time)` to schedule something to be run a specific time.

## Example

This example runs my\_Procedure 15 seconds from now.

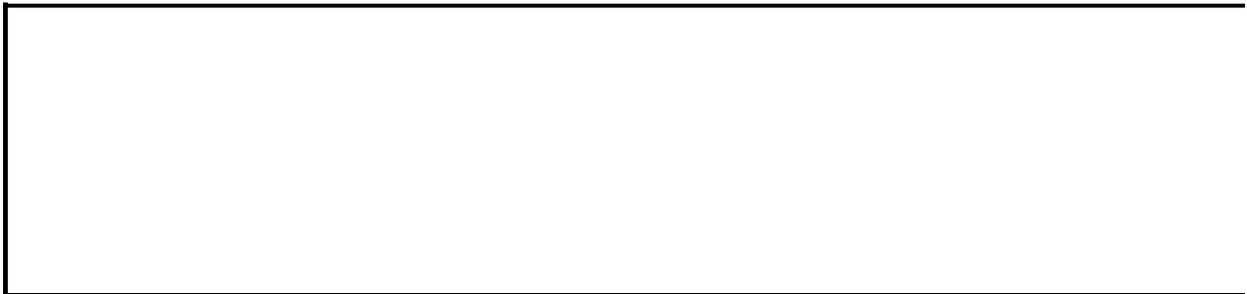
```
Application.OnTime Now + TimeValue("00:00:15"), "my_Procedure"
```

This example runs my\_Procedure at 5 P.M.

```
Application.OnTime TimeValue("17:00:00"), "my_Procedure"
```

This example cancels the **OnTime** setting from the previous example.

```
Application.OnTime EarliestTime:=TimeValue("17:00:00"), _  
    Procedure:="my_Procedure", Schedule:=False
```



# OnUndo Method

Sets the text of the **Undo** and the name of the procedure that's run if you choose the **Undo** command (**Edit** menu) after running the procedure that sets this property.

*expression.OnUndo(Text, Procedure)*

*expression* Required. An expression that returns an **Application** object.

**Text** Required **String**. The text that appears with the **Undo** command (**Edit** menu).

**Procedure** Required **String**. The name of the procedure that's run when you choose the **Undo** command (**Edit** menu).

## Remarks

If a procedure doesn't use the **OnUndo** method, the **Undo** command is disabled.

The procedure must use the **OnRepeat** and **OnUndo** methods last, to prevent the repeat and undo procedures from being overwritten by subsequent actions in the procedure.

## Example

This example sets the repeat and undo procedures.

```
Application.OnRepeat "Repeat VB Procedure", _  
    "Book1.xls!My_Repeat_Sub"  
Application.OnUndo "Undo VB Procedure", _  
    "Book1.xls!My_Undo_Sub"
```



[Show All](#)

# Open Method

 [Open method as it applies to the \*\*Workbooks\*\* object.](#)

Opens a workbook.

*expression*.**Open**(*FileName*, *UpdateLinks*, *ReadOnly*, *Format*, *Password*, *WriteResPassword*, *IgnoreReadOnlyRecommended*, *Origin*, *Delimiter*, *Editable*, *Notify*, *Converter*, *AddToMru*, *Local*, *CorruptLoad*)

*expression* Required. An expression that returns the **Workbooks** object.

**FileName** Required **String**. The file name of the workbook to be opened.

**UpdateLinks** Optional **VARIANT**. Specifies the way links in the file are updated. If this argument is omitted, the user is prompted to specify how links will be updated. Otherwise, this argument is one of the values listed in the following table.

Value	Meaning
0	Doesn't update any references
1	Updates external references but not remote references
2	Updates remote references but not external references
3	Updates both remote and external references

If Microsoft Excel is opening a file in the WKS, WK1, or WK3 format and the **UpdateLinks** argument is 2, Microsoft Excel generates charts from the graphs attached to the file. If the argument is 0, no charts are created.

**ReadOnly** Optional **VARIANT**. True to open the workbook in read-only mode.

**Format** Optional **VARIANT**. If Microsoft Excel is opening a text file, this argument specifies the delimiter character, as shown in the following table. If this argument is omitted, the current delimiter is used.

Value	Delimiter
-------	-----------

- 1 Tabs
- 2 Commas
- 3 Spaces
- 4 Semicolons
- 5 Nothing
- 6 Custom character (see the ***Delimiter*** argument)

***Password*** Optional **Vari**ant. A string that contains the password required to open a protected workbook. If this argument is omitted and the workbook requires a password, the user is prompted for the password.

***WriteResPassword*** Optional **Vari**ant. A string that contains the password required to write to a write-reserved workbook. If this argument is omitted and the workbook requires a password, the user will be prompted for the password.

***IgnoreReadOnlyRecommended*** Optional **Vari**ant. **True** to have Microsoft Excel not display the read-only recommended message (if the workbook was saved with the **Read-Only Recommended** option).

***Origin*** Optional **Vari**ant. If the file is a text file, this argument indicates where it originated (so that code pages and Carriage Return/Line Feed (CR/LF) can be mapped correctly). Can be one of the following **XIPlatform** constants: **xlMacintosh**, **xlWindows**, or **xlMSDOS**. If this argument is omitted, the current operating system is used.

***Delimiter*** Optional **Vari**ant. If the file is a text file and the **Format** argument is 6, this argument is a string that specifies the character to be used as the delimiter. For example, use Chr(9) for tabs, use "," for commas, use ";" for semicolons, or use a custom character. Only the first character of the string is used.

***Editable*** Optional **Vari**ant. If the file is a Microsoft Excel 4.0 add-in, this argument is **True** to open the add-in so that it's a visible window. If this argument is **False** or omitted, the add-in is opened as hidden, and it cannot be unhidden. This option doesn't apply to add-ins created in Microsoft Excel 5.0 or later. If the file is an Excel template, **True** to open the specified template for editing. **False** to open a new workbook based on the specified template. The default value is **False**.

***Notify*** Optional **Vari**ant. If the file cannot be opened in read/write mode, this

argument is **True** to add the file to the file notification list. Microsoft Excel will open the file as read-only, poll the file notification list, and then notify the user when the file becomes available. If this argument is **False** or omitted, no notification is requested, and any attempts to open an unavailable file will fail.

**Converter** Optional **Variant**. The index of the first file converter to try when opening the file. The specified file converter is tried first; if this converter doesn't recognize the file, all other converters are tried. The converter index consists of the row numbers of the converters returned by the [FileConverters](#) property.

**AddToMru** Optional **Variant**. **True** to add this workbook to the list of recently used files. The default value is **False**.

**Local** Optional **Variant**. **True** saves files against the language of Microsoft Excel (including control panel settings). **False** (default) saves files against the language of [Visual Basic for Applications \(VBA\)](#) (which is typically US English unless the VBA project where Workbooks.Open is run from is an old internationalized XL5/95 VBA project).

**CorruptLoad** Optional **Variant**. Can be one of the following constants: **xlNormalLoad**, **xlRepairFile** and **xlExtractData**. The Default behavior if no value is specified is usually normal but may be safe load or data recovery, if Excel has already attempted to open the file. The first attempt is normal. If Excel stops operating while opening the file the second attempt is safe load. If Excel again stops operating the next attempt is data recovery.



[Open method as it applies to the \*\*RecentFile\*\* object.](#)

Opens a recent workbook.

*expression*.**Open**

*expression* Required. An expression that returns the **RecentFile** object.

## Example

This example opens the workbook Analysis.xls and then runs its Auto\_Open macro.

```
Workbooks.Open "ANALYSIS.XLS"  
ActiveWorkbook.RunAutoMacros xlAutoOpen
```



# OpenDatabase Method

Returns a **Workbook** object representing a database.

*expression*.**OpenDatabase**(*FileName*, *CommandText*, *CommandType*, *BackgroundQuery*, *ImportDataAs*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**FileName** Required **String**. The connection string.

**CommandText** Optional **Variant**. The command text of the query.

**CommandType** Optional **Variant**. The command type of the query. The following command types are available: Default, SQL, and Table.

**BackgroundQuery** Optional **Variant**. The background of the query.

**ImportDataAs** Optional **Variant**. Determines the format of the query.

## Example

In this example, Microsoft Excel opens the "northwind.mdb" file. This example assumes a file called "northwind.mdb file" exists on the C:\ drive.

```
Sub UseOpenDatabase()  
    ' Open the Northwind database.  
    Workbooks.OpenDatabase _  
        FileName:="C:\northwind.mdb"  
End Sub
```



[Show All](#)

# OpenLinks Method

Opens the supporting documents for a link or links.

*expression*.**OpenLinks**(*Name*, *ReadOnly*, *Type*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Name** Required **String**. The name of the Microsoft Excel or DDE/OLE link, as returned from the [LinkSources](#) method.

**ReadOnly** Optional **Variant**. **True** to open documents as read-only. The default value is **False**.

**Type** Optional [XLink](#). The link type.

XLink can be one of these XLink constants.

**xlExcelLinks**

**xlOLELinks** (also handles DDE links)

**xlPublishers**

**xlSubscribers**

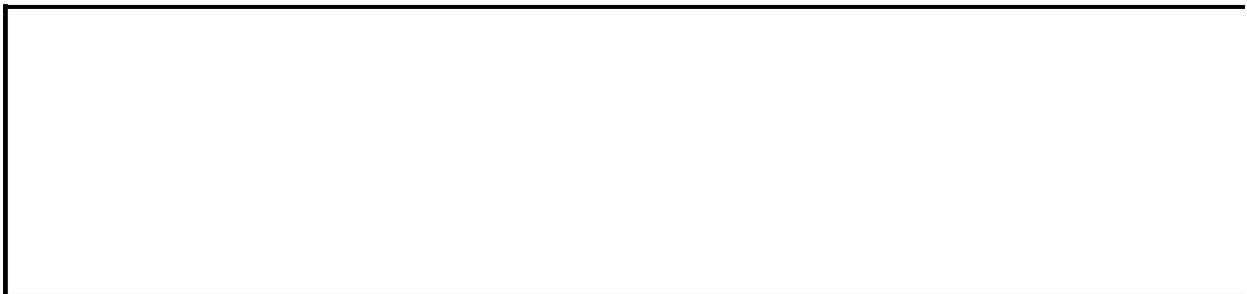
## Example

This example opens OLE link one in the active workbook.

```
linkArray = ActiveWorkbook.LinkSources(xlOLELinks)  
ActiveWorkbook.OpenLinks linkArray(1)
```

This example opens all supporting Microsoft Excel documents for the active workbook.

```
Sub OpenAllLinks()  
    Dim arLinks As Variant  
    Dim intIndex As Integer  
    arLinks = ActiveWorkbook.LinkSources(xlExcelLinks)  
  
    If Not IsEmpty(arLinks) Then  
        For intIndex = LBound(arLinks) To UBound(arLinks)  
            ActiveWorkbook.OpenLinks arLinks(intIndex)  
        Next intIndex  
    Else  
        MsgBox "The active workbook contains no external links."  
    End If  
End Sub
```



[Show All](#)

# OpenText Method

Loads and parses a text file as a new workbook with a single sheet that contains the parsed text-file data.

*expression.OpenText(FileName, Origin, StartRow, DataType, TextQualifier, ConsecutiveDelimiter, Tab, Semicolon, Comma, Space, Other, OtherChar, FieldInfo, TextVisualLayout, DecimalSeparator, ThousandsSeparator, TrailingMinusNumbers, Local)*

*expression* Required. An expression that returns one of the objects in the Applies To list.

**FileName** Required **String**. Specifies the file name of the text file to be opened and parsed.

**Origin** Optional **Variant**. Specifies the origin of the text file. Can be one of the following **XIPlatform** constants: **xlMacintosh**, **xlWindows**, or **xlMSDOS**. Additionally, this could be an integer representing the code page number of the desired code page. For example, "1256" would specify that the encoding of the source text file is Arabic (Windows). If this argument is omitted, the method uses the current setting of the **File Origin** option in the **Text Import Wizard**.

**StartRow** Optional **Variant**. The row number at which to start parsing text. The default value is 1.

**DataType** Optional **Variant**. Specifies the column format of the data in the file. Can be one of the following **XITextParsingType** constants: **xlDelimited** or **xlFixedWidth**. If this argument is not specified, Microsoft Excel attempts to determine the column format when it opens the file.

**TextQualifier** Optional [XITextQualifier](#). Specifies the text qualifier.

XITextQualifier can be one of these XITextQualifier constants.

**xlTextQualifierDoubleQuote** *default*

**xlTextQualifierNone**

**xlTextQualifierSingleQuote**

**ConsecutiveDelimiter** Optional **VARIANT**. **True** to have consecutive delimiters considered one delimiter. The default is **False**.

**Tab** Optional **VARIANT**. **True** to have the tab character be the delimiter (**DataType** must be **xlDelimited**). The default value is **False**.

**Semicolon** Optional **VARIANT**. **True** to have the semicolon character be the delimiter (**DataType** must be **xlDelimited**). The default value is **False**.

**Comma** Optional **VARIANT**. **True** to have the comma character be the delimiter (**DataType** must be **xlDelimited**). The default value is **False**.

**Space** Optional **VARIANT**. **True** to have the space character be the delimiter (**DataType** must be **xlDelimited**). The default value is **False**.

**Other** Optional **VARIANT**. **True** to have the character specified by the **OtherChar** argument be the delimiter (**DataType** must be **xlDelimited**). The default value is **False**.

**OtherChar** Optional **VARIANT** (required if **Other** is **True**). Specifies the delimiter character when **Other** is **True**. If more than one character is specified, only the first character of the string is used; the remaining characters are ignored.

**FieldInfo** Optional **xlColumnDataType**. An array containing parse information for individual columns of data. The interpretation depends on the value of **DataType**. When the data is delimited, this argument is an array of two-element arrays, with each two-element array specifying the conversion options for a particular column. The first element is the column number (1-based), and the second element is one of the [xlColumnDataType](#) constants specifying how the column is parsed.

**xlColumnDataType** can be one of these **xlColumnDataType** constants.

**xlGeneralFormat** General

**xlTextFormat** Text

**xlMDYFormat** MDY date

**xlDMYFormat** DMY date

**xlYMDFormat** YMD date

**xlMYDFormat** MYD date

**xlDYMFormat** DYM date

**xlYDMFormat** YDM date

**xlEMDFormat** EMD date

**xlSkipColumn** Skip Column

You can use **xlEMDFormat** only if you have installed and selected Taiwanese language support. The **xlEMDFormat** constant specifies that Taiwanese era dates are being used.

The column specifiers can be in any order. If there's no column specifier for a particular column in the input data, the column is parsed with the General setting.

## Notes

- If you specify that a column is to be skipped, you must explicitly state the type for all remaining columns or the data will not parse correctly.
- If there is a recognizable date in the data, the cell will be formatted as a date in the worksheet even if the setting for the column is General. Additionally, if you specify one of the above date formats for a column and the data does not contain a recognized date, then the cell format in the worksheet will be General.

This example causes the third column to be parsed as MDY (for example, 01/10/1970), the first column to be parsed as text, and the remaining columns in the source data to be parsed with the General setting.

```
Array(Array(3, 3), Array(1, 2))
```

If the source data has fixed-width columns, the first element in each two-element array specifies the position of the starting character in the column (as an integer; character 0 (zero) is the first character). The second element in the two-element

array specifies the parse option for the column as a number between 0 and 9, as listed in the preceding table.

**TextVisualLayout** Optional **Variant**. The visual layout of the text.

**DecimalSeparator** Optional **Variant**. The decimal separator that Microsoft Excel uses when recognizing numbers. The default setting is the system setting.

**ThousandsSeparator** Optional **Variant**. The thousands separator that Excel uses when recognizing numbers. The default setting is the system setting.

The following [table](#) shows the results of importing text into Excel for various import settings. Numeric results are displayed in the rightmost column.

<b>System decimal separator</b>	<b>System thousands separator</b>	<b>Decimal separator value</b>	<b>Thousands separator value</b>	<b>Text imported</b>	<b>Cell value (data type)</b>
Period	Comma	Comma	Period	123.123,45	123,123.45 (numeric)
Period	Comma	Comma	Comma	123.123,45	123.123,45 (text)
Comma	Period	Period	Comma	123,123.45	123,123.45 (numeric)
Period	Comma	Period	Comma	123 123.45	123 123.45 (text)
Period	Comma	Period	Space	123 123.45	123,123.45 (numeric)

**TrailingMinusNumbers** Optional **Variant**.

**Local** Optional **Variant**.

## Example

This example opens the file Data.txt and uses tab delimiters to parse the text file into a worksheet.

```
Workbooks.OpenText filename:="DATA.TXT", _  
    dataType:=xlDelimited, tab:=True
```



[Show All](#)

# OpenXML Method

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Opens an XML data file. Returns a [Workbook](#) object.

*expression*.**OpenXML**(*Filename*, *Stylesheets*, *LoadOption*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**FileName** Required **String**. The name of the file to open.

**Stylesheets** Optional **Variant**. Either a single value or an array of values that specify which [XSL Transformation \(XSLT\)](#) stylesheet processing instructions to apply.

**LoadOption** Optional **Variant**. Specifies how Excel opens the XML data file. Can be one of the [XIxmlLoadOption](#) constants.

**XIxmlLoadOption** can be one of these **XIxmlLoadOption** constants:

**xlXmlLoadImportToList** Places the contents of the XML data file in an XML list.

**xlXmlLoadMapXml** Displays the schema of the XML data file in the **XML Structure** task pane.

**xlXmlLoadOpenXml** Opens the XML data file. The contents of the file will be flattened.

**xlXmlLoadPromptUser** Prompts the user to choose how to open the file.

## Example

The following code opens the XML data file "customers.xml" and displays the file's contents in an XML list.

```
Sub UseOpenXML()  
    Application.Workbooks.OpenXML _  
        Filename:="customers.xml", _  
        LoadOption:=xlXmlLoadImportToList  
End Sub
```



# Parse Method

Parses a range of data and breaks it into multiple cells. Distributes the contents of the range to fill several adjacent columns; the range can be no more than one column wide.

*expression*.**Parse**(*ParseLine*, *Destination*)

*expression* Required. An expression that returns a **Range** object.

**ParseLine** Optional **Variant**. A string that contains left and right brackets to indicate where the cells should be split. For example, "[xxx][xxx]" would insert the first three characters into the first column of the destination range, and it would insert the next three characters into the second column. If this argument is omitted, Microsoft Excel guesses where to split the columns based on the spacing of the top left cell in the range. If you want to use a different range to guess the parse line, use a **Range** object as the **ParseLine** argument. That range must be one of the cells that's being parsed. The **ParseLine** argument cannot be longer than 255 characters, including the brackets and spaces.

**Destination** Optional **Variant**. A **Range** object that represents the upper-left corner of the destination range for the parsed data. If this argument is omitted, Microsoft Excel parses in place.

## Example

This example divides telephone numbers of the form 206-555-1212 into two columns. The first column contains only the area code, and the second column contains the seven-digit telephone number with the embedded hyphen.

```
Worksheets("Sheet1").Columns("A").Parse _  
    parseLine:="[xxx] [xxxxxxx]", _  
    destination:=Worksheets("Sheet1").Range("B1")
```



[Show All](#)

# Paste Method

 [Paste method as it applies to the \*\*Chart\*\* object.](#)

Pastes chart data from the Clipboard into the specified chart.

*expression*.**Paste**(*Type*)

*expression* Required. An expression that returns a [Chart](#) object

**Type** Optional **Variant**. Specifies the chart information to paste if a chart is on the Clipboard. Can be one of the following **XlPasteType** constants: **xlFormats**, **xlFormulas**, or **xlAll**. The default value is **xlAll**. If there's data other than a chart on the Clipboard, this argument cannot be used.

## Remark

This method changes the current selection.

 [Paste method as it applies to the \*\*Floor\*\*, \*\*Point\*\*, \*\*Series\*\*, and \*\*Walls\*\* objects.](#)

For **Floor** and **Walls** objects, paste a picture from the Clipboard on the floor or walls of the specified chart. For **Point** and **Series** objects, pastes a picture from the Clipboard as the marker on the selected point or series. This method can be used on column, bar, line, or radar charts, and it sets the **MarkerStyle** property to **xlMarkerStylePicture**.

*expression.Paste*

*expression* Required. An expression that returns one of the above objects.

 [Paste method as it applies to the \*\*SeriesCollection\*\* object.](#)

Pastes data from the Clipboard into the specified series collection.

*expression.Paste(Rowcol, SeriesLabels, CategoryLabels, Replace, NewSeries)*

*expression* Required. An expression that returns a [SeriesCollection](#) object.

**Rowcol** Optional [XlRowCol](#). Specifies whether the values corresponding to a particular data series are in rows or columns.

XlRowCol can be one of these XlRowCol constants.

**xlColumns** *default*

**xlRows**

**SeriesLabels** Optional **Variant**. **True** to use the contents of the cell in the first column of each row (or the first row of each column) as the name of the data series in that row (or column). **False** to use the contents of the cell in the first column of each row (or the first row of each column) as the first data point in the data series. The default value is **False**.

**CategoryLabels** Optional **Variant**. **True** to use the contents of the first row (or column) of the selection as the categories for the chart. **False** to use the contents of the first row (or column) as the first data series in the chart. The default value is **False**.

**Replace** Optional **Variant**. **True** to apply categories while replacing existing categories with information from the copied range. **False** to insert new categories without replacing any old ones. The default value is **True**.

**NewSeries** Optional **Variant**. **True** to paste the data as a new series. **False** to paste the data as new points in an existing series. The default value is **True**.



[Paste method as it applies to the \*\*Worksheet\*\* object.](#)

Pastes the contents of the Clipboard onto the sheet.

*expression*.**Paste**(*Destination*, *Link*)

*expression* Required. An expression that returns a [Worksheet](#) object.

**Destination** Optional **Variant**. A **Range** object that specifies where the Clipboard contents should be pasted. If this argument is omitted, the current selection is used. This argument can be specified only if the contents of the Clipboard can be pasted into a range. If this argument is specified, the **Link** argument cannot be used.

**Link** Optional **Variant**. **True** to establish a link to the source of the pasted data. If this argument is specified, the **Destination** argument cannot be used. The default value is **False**.

## Remarks

If you don't specify the ***Destination*** argument, you must select the destination range before you use this method.

This method may modify the sheet selection, depending on the contents of the Clipboard.

## Example

[As it applies to the \*\*Chart\*\* object.](#)

This example pastes data from the range B1:B5 on Sheet1 into Chart1.

```
Worksheets("Sheet1").Range("B1:B5").Copy  
Charts("Chart1").Paste
```

[As it applies to the \*\*Point\*\* or \*\*Series\*\* objects.](#)

This example pastes a picture from the Clipboard into series one in Chart1.

```
Charts("Chart1").SeriesCollection(1).Paste
```

[As it applies to the \*\*SeriesCollection\*\* object.](#)

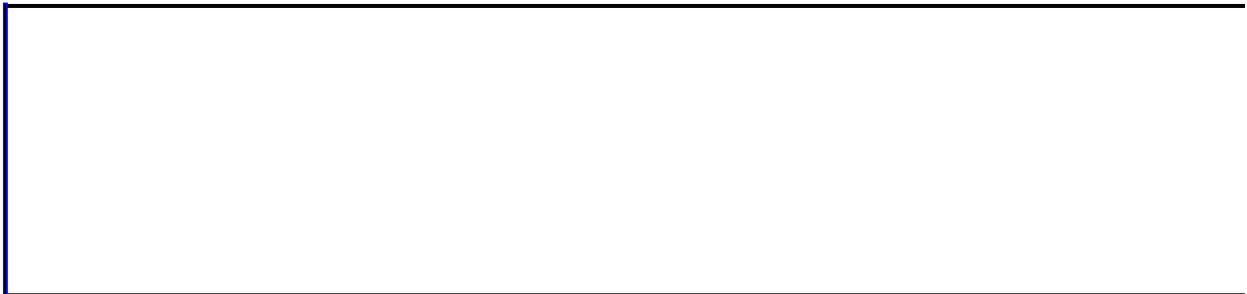
This example pastes a picture from the Clipboard into series one in Chart1.

```
Worksheets("Sheet1").Range("C1:C5").Copy  
Charts("Chart1").SeriesCollection.Paste
```

[As it applies to the \*\*Worksheet\*\* object.](#)

This example copies data from cells C1:C5 on Sheet1 to cells D1:D5 on Sheet1.

```
Worksheets("Sheet1").Range("C1:C5").Copy  
ActiveSheet.Paste Destination:=Worksheets("Sheet1").Range("D1:D5")
```



[Show All](#)

# PasteSpecial Method

 [PasteSpecial method as it applies to the Range object.](#)

Pastes a **Range** from the Clipboard into the specified range.

*expression*.**PasteSpecial**(*Paste*, *Operation*, *SkipBlanks*, *Transpose*)

*expression* Required. An expression that returns a **Range** object.

*Paste* Optional [XlPasteType](#). The part of the range to be pasted.

XlPasteType can be one of these XlPasteType constants.

**xlPasteAll** *default*

**xlPasteAllExceptBorders**

**xlPasteColumnWidths**

**xlPasteComments**

**xlPasteFormats**

**xlPasteFormulas**

**xlPasteFormulasAndNumberFormats**

**xlPasteValidation**

**xlPasteValues**

**xlPasteValuesAndNumberFormats**

*Operation* Optional [XlPasteSpecialOperation](#). The paste operation.

XlPasteSpecialOperation can be one of these XlPasteSpecialOperation constants.

**xlPasteSpecialOperationAdd**

**xlPasteSpecialOperationDivide**

**xlPasteSpecialOperationMultiply**

**xlPasteSpecialOperationNone** *default*

**xlPasteSpecialOperationSubtract**

**SkipBlanks** Optional **Variant**. **True** to have blank cells in the range on the Clipboard not be pasted into the destination range. The default value is **False**.

**Transpose** Optional **Variant**. **True** to transpose rows and columns when the range is pasted. The default value is **False**.

[PasteSpecial method as it applies to the \*\*Worksheet\*\* object.](#)

Pastes the contents of the Clipboard onto the sheet, using a specified format. Use this method to paste data from other applications or to paste data in a specific format.

*expression*.**PasteSpecial**(*Format, Link, DisplayAsIcon, IconFileName, IconIndex, IconLabel, NoHTMLFormatting*)

*expression* Required. An expression that returns a **Worksheet** object.

**Format** Optional **Variant**. A string that specifies the Clipboard format of the data.

**Link** Optional **Variant**. **True** to establish a link to the source of the pasted data. If the source data isn't suitable for linking or the source application doesn't support linking, this parameter is ignored. The default value is **False**.

**DisplayAsIcon** Optional **Variant**. **True** to display the pasted as an icon. The default value is **False**.

**IconFileName** Optional **Variant**. The name of the file that contains the icon to use if **DisplayAsIcon** is **True**.

**IconIndex** Optional **Variant**. The index number of the icon within the icon file.

**IconLabel** Optional **Variant**. The text label of the icon.

**NoHTMLFormatting** Optional **Variant**. **True** to remove all formatting, hyperlinks, and images from HTML. **False** to paste HTML as is. The default value is **False**.

## Remarks

**Note** *NoHTMLFormatting* will only matter when *Format* = “HTML”. In all other cases, *NoHTMLFormatting* will be ignored.

You must select the destination range before you use this method.

This method may modify the sheet selection, depending on the contents of the Clipboard.

## Example

[As it applies to the \*\*Range\*\* object.](#)

This example replaces the data in cells D1:D5 on Sheet1 with the sum of the existing contents and cells C1:C5 on Sheet1.

```
With Worksheets("Sheet1")
    .Range("C1:C5").Copy
    .Range("D1:D5").PasteSpecial _
        Operation:=xlPasteSpecialOperationAdd
End With
```

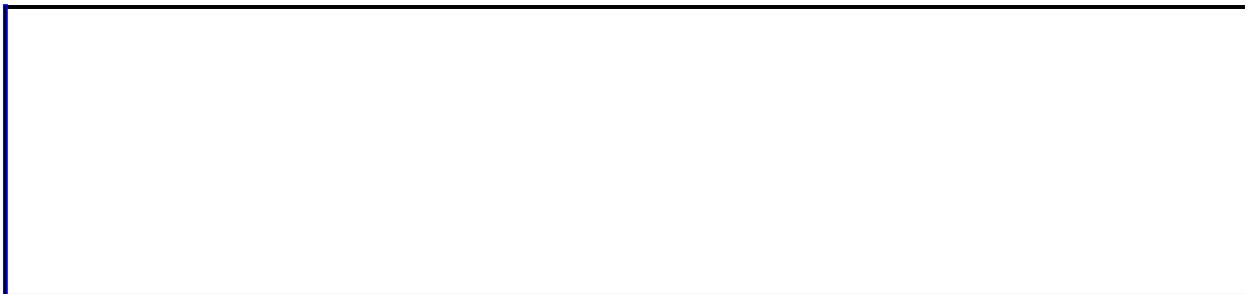
[As it applies to the \*\*Worksheet\*\* object.](#)

This example pastes a Microsoft Word document object from the Clipboard to cell D1 on Sheet1.

```
Worksheets("Sheet1").Range("D1").Select
ActiveSheet.PasteSpecial format:= _
    "Microsoft Word 8.0 Document Object"
```

This example pastes the same Microsoft Word document object and displays it as an icon.

```
Worksheets("Sheet1").Range("F5").Select
ActiveSheet.PasteSpecial _
    Format:="Microsoft Word 8.0 Document Object", _
    DisplayAsIcon:=True
```



[Show All](#)

# Patterned Method

Sets the specified fill to a pattern.

*expression*.**Patterned**(*Pattern*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*Pattern* Required [MsoPatternType](#).

MsoPatternType can be one of these MsoPatternType constants.

**msoPattern10Percent**

**msoPattern20Percent**

**msoPattern25Percent**

**msoPattern30Percent**

**msoPattern40Percent**

**msoPattern50Percent**

**msoPattern5Percent**

**msoPattern60Percent**

**msoPattern70Percent**

**msoPattern75Percent**

**msoPattern80Percent**

**msoPattern90Percent**

**msoPatternDarkDownwardDiagonal**

**msoPatternDarkHorizontal**

**msoPatternDarkUpwardDiagonal**

**msoPatternDarkVertical**

**msoPatternDashedDownwardDiagonal**

**msoPatternDashedHorizontal**

**msoPatternDashedUpwardDiagonal**

**msoPatternDashedVertical**

**msoPatternDiagonalBrick**

**msoPatternDivot**  
**msoPatternDottedDiamond**  
**msoPatternDottedGrid**  
**msoPatternHorizontalBrick**  
**msoPatternLargeCheckerBoard**  
**msoPatternLargeConfetti**  
**msoPatternLargeGrid**  
**msoPatternLightDownwardDiagonal**  
**msoPatternLightHorizontal**  
**msoPatternLightUpwardDiagonal**  
**msoPatternLightVertical**  
**msoPatternMixed**  
**msoPatternNarrowHorizontal**  
**msoPatternNarrowVertical**  
**msoPatternOutlinedDiamond**  
**msoPatternPlaid**  
**msoPatternShingle**  
**msoPatternSmallCheckerBoard**  
**msoPatternSmallConfetti**  
**msoPatternSmallGrid**  
**msoPatternSolidDiamond**  
**msoPatternSphere**  
**msoPatternTrellis**  
**msoPatternWave**  
**msoPatternWeave**  
**msoPatternWideDownwardDiagonal**  
**msoPatternWideUpwardDiagonal**  
**msoPatternZigZag**

## Example

This example sets the fill pattern for chart one.

```
With Charts(1).ChartArea.Fill  
    .Patterned msoPatternDiagonalBrick  
    .Visible = True  
End With
```



# PickUp Method

Copies the formatting of the specified shape. Use the [Apply](#) method to apply the copied formatting to another shape.

*expression*.**PickUp**

*expression* Required. An expression that returns a **Shape** or **ShapeRange** object.

## Example

This example copies the formatting of shape one on myDocument and then applies the copied formatting to shape two.

```
Set myDocument = Worksheets(1)
With myDocument
    .Shapes(1).PickUp
    .Shapes(2).Apply
End With
```



# PieGroups Method

On a 2-D chart, returns an object that represents either a single pie chart group (a [ChartGroup](#) object) or a collection of the pie chart groups (a [ChartGroups](#) collection).

*expression*.**PieGroups**(*Index*)

*expression* Required. An expression that returns a **Chart** object.

*Index* Optional **Variant**. Specifies the chart group.

## Example

This example sets pie group one in Chart1 to use a different color for each data marker. The example should be run on a 2-D chart.

```
Charts("Chart1").PieGroups(1).VaryByCategories = True
```



# PivotCache Method

Returns a [PivotCache](#) object that represents the cache for the specified PivotTable report. Read-only.

*expression*.**PivotCache**

*expression* Required. An expression that returns a **PivotTable** object.

## Example

This example causes the PivotTable cache for the first PivotTable report on worksheet one to be optimized when it's constructed.

```
Worksheets(1).PivotTables("Pivot1") _  
    .PivotCache.OptimizeCache = True
```



# PivotCaches Method

Returns a [PivotCaches](#) collection that represents all the PivotTable caches in the specified workbook. Read-only.

*expression*.**PivotCaches**

*expression* Required. An expression that returns a **Workbook** object.

## Example

This example causes the PivotTable cache to update automatically each time the workbook is opened.

```
ActiveWorkbook.PivotCaches(1).RefreshOnFileOpen = True
```



[Show All](#)

# PivotFields Method

Returns an object that represents either a single PivotTable field (a [PivotField](#) object) or a collection of both the visible and hidden fields (a [PivotFields](#) object) in the PivotTable report. Read-only.

*expression*.**PivotFields**(*Index*)

*expression* Required. An expression that returns a **PivotTable** object.

*Index* Optional **Variant**. The name or number of the field to be returned.

## Remarks

For [OLAP](#) data sources, there are no hidden fields, and the object or collection that's returned reflects what's currently visible.

## Example

This example adds the PivotTable report's field names to a list on a new worksheet.

```
Set nwSheet = Worksheets.Add  
nwSheet.Activate  
Set pvtTable = Worksheets("Sheet2").Range("A1").PivotTable  
rw = 0  
For Each pvtField In pvtTable.PivotFields  
    rw = rw + 1  
    nwSheet.Cells(rw, 1).Value = pvtField.Name  
Next pvtField
```



[Show All](#)

# PivotItems Method

Returns an object that represents either a single PivotTable item (a [PivotItem](#) object) or a collection of all the visible and hidden items (a [PivotItems](#) object) in the specified field. Read-only.

*expression*.**PivotItems**(*Index*)

*expression* Required. An expression that returns a **PivotField** object.

*Index* Optional **Variant**. The name or number of the item to be returned.

## Remarks

For [OLAP](#) data sources, the collection is indexed by the unique name (the name returned by the [SourceName](#) property), not by the display name.

## Example

This example adds the names of all items in the field named "product" to a list on a new worksheet.

```
Set nwSheet = Worksheets.Add  
nwSheet.Activate  
Set pvtTable = Worksheets("Sheet2").Range("A1").PivotTable  
rw = 0  
For Each pvtitem In pvtTable.PivotFields("product").PivotItems  
    rw = rw + 1  
    nwSheet.Cells(rw, 1).Value = pvtitem.Name  
Next
```



[Show All](#)

# PivotSelect Method

Selects part of a PivotTable report.

*expression*.**PivotSelect**(*Name*, *Mode*, *UseStandardName*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Name** Required **String**. The selection, in standard PivotTable report selection format.

**Mode** Optional [XIPTSelectionMode](#). Specifies the structured selection mode.

XIPTSelectionMode can be one of these XIPTSelectionMode constants.

**xlBlanks**

**xlButton**

**xlDataAndLabel** *default*

**xlDataOnly**

**xlFirstRow**

**xlLabelOnly**

**xlOrigin**

**UseStandardName** Optional **Variant**. **True** for recorded macros that will play back in other locales.

## Remarks

You can use the specified mode only to select the corresponding item in the PivotTable report. For example, you cannot select data and labels by using **xlButton** mode; likewise, you cannot select buttons by using **xlDataOnly** mode.

## Example

This example selects all date labels in the first PivotTable report on worksheet one.

```
Worksheets(1).PivotTables(1).PivotSelect "date[All]", xlLabelOnly
```



# PivotTables Method

Returns an object that represents either a single PivotTable report (a [PivotTable](#) object) or a collection of all the PivotTable reports (a [PivotTables](#) object) on a worksheet. Read-only.

*expression*.**PivotTables**(*Index*)

*expression* Required. An expression that returns a **Worksheet** object.

*Index* Optional **VARIANT**. The name or number of the report.

## Example

This example sets the Sum of 1994 field in the first PivotTable report on the active sheet to use the SUM function.

```
ActiveSheet.PivotTables("PivotTable1").  
    PivotFields("Sum of 1994").Function = xlSum
```



[Show All](#)

# PivotTableWizard Method

 [PivotTableWizard method as it applies to the \*\*Worksheet\*\* object.](#)

Creates a **PivotTable** object. This method doesn't display the PivotTable Wizard. This method isn't available for OLE DB data sources. Use the **Add** method to add a PivotTable cache, and then create a PivotTable report based on the cache. **PivotTable** object.

*expression.PivotTableWizard(SourceType, SourceData, TableDestination, TableName, RowGrand, ColumnGrand, SaveData, HasAutoFormat, AutoPage, Reserved, BackgroundQuery, OptimizeCache, PageFieldOrder, PageFieldWrapCount, ReadData, Connection)*

*expression* Required. An expression that returns one of the above objects.

**SourceType** Optional [XlPivotTableSourceType](#). The source of the report data.

XlPivotTableSourceType can be one of these XlPivotTableSourceType constants.

**xlConsolidation**. Multiple consolidation ranges

**xlDatabase**. Microsoft Excel list or database

**xlExternal**. Data from another application

**xlPivotTable**. Same source as another PivotTable report

If you specify this argument, you must also specify **SourceData**. If **SourceType** and **SourceData** are omitted, Microsoft Excel assumes that the source type is **xlDatabase**, and the source data comes from the named range "Database." If this named range doesn't exist, Microsoft Excel uses the current region if the current selection is in a range of more than 10 cells that contain data. If this isn't true, this method will fail.

**SourceData** Optional **Variant**. The data for the new report. Can be a [Range](#) object, an array of ranges, or a text constant that represents the name of another

report. For an external database, **SourceData** is an array of strings containing the SQL query string, where each element is up to 255 characters in length. You should use the **Connection** argument to specify the ODBC connection string. For compatibility with earlier versions of Excel, **SourceData** can be a two-element array. The first element is the connection string specifying the ODBC source for the data. The second element is the SQL query string used to get the data. If you specify **SourceData**, you must also specify **SourceType**. If the active cell is inside the **SourceData** range, you must specify **TableDestination** as well.

**TableDestination** Optional **Variant**. A **Range** object specifying where the report should be placed on the worksheet. If this argument is omitted, the report is placed at the active cell.

**TableName** Optional **Variant**. A string that specifies the name of the new report.

**RowGrand** Optional **Variant**. **True** to show grand totals for rows in the report.

**ColumnGrand** Optional **Variant**. **True** to show grand totals for columns in the report.

**SaveData** Optional **Variant**. **True** to save data with the report. **False** to save only the report definition.

**HasAutoFormat** Optional **Variant**. **True** to have Microsoft Excel automatically format the report when it's refreshed or when fields are moved.

**AutoPage** Optional **Variant**. Valid only if **SourceType** is **xlConsolidation**. **True** to have Microsoft Excel create a page field for the consolidation. If **AutoPage** is **False**, you must create the page field or fields.

**Reserved** Optional **Variant**. Not used by Microsoft Excel.

**BackgroundQuery** Optional **Variant**. **True** to have Excel perform queries for the report asynchronously (in the background). The default value is **False**.

**OptimizeCache** Optional **Variant**. **True** to optimize the PivotTable cache when it's constructed. The default value is **False**.

**PageFieldOrder** Optional **Variant**. The order in which page fields are added

to the PivotTable report's layout. Can be one of the following **XIOrder** constants: **xlDownThenOver** or **xlOverThenDown**. The default value is **xlDownThenOver**.

**PageFieldWrapCount** Optional **Variant**. The number of page fields in each column or row in the PivotTable report. The default value is 0 (zero).

**ReadData** Optional **Variant**. **True** to create a PivotTable cache that contains all records from the external database; this cache can be very large. If **ReadData** is **False**, you can set some of the fields as server-based page fields before the data is actually read.

**Connection** Optional **Variant**. A string that contains ODBC settings that allow Excel to connect to an ODBC data source. The connection string has the form "ODBC;<connection string>". This argument overrides any previous setting for the **PivotCache** object's **Connection** property.

 [PivotTableWizard method as it applies to the \*\*PivotTable\*\* and \*\*Workbook\*\* objects.](#)

Creates a **PivotTable** object. This method doesn't display the PivotTable Wizard. This method isn't available for OLE DB data sources. Use the **Add** method to add a PivotTable cache, and then create a PivotTable report based on the cache.

*expression.PivotTableWizard(SourceType, SourceData, TableDestination, TableName, RowGrand, ColumnGrand, SaveData, HasAutoFormat, AutoPage, Reserved, BackgroundQuery, OptimizeCache, PageFieldOrder, PageFieldWrapCount, ReadData, Connection)*

*expression* Required. An expression that returns one of the above objects.

**SourceType** Optional **XIPivotTableSourceType**. The source of the report data.

XIPivotTableSourceType can be one of these XIPivotTableSourceType constants.

**xlConsolidation**. Multiple consolidation ranges

**xlDatabase**. Microsoft Excel list or database

**xlExternal**. Data from another application

**xlPivotTable.** Same source as another PivotTable report

If you specify this argument, you must also specify **SourceData**. If **SourceType** and **SourceData** are omitted, Microsoft Excel assumes that the source type is **xlDatabase**, and the source data comes from the named range "Database." If this named range doesn't exist, Microsoft Excel uses the current region if the current selection is in a range of more than 10 cells that contain data. If this isn't true, this method will fail.

**SourceData** Optional **Variant**. The data for the new report. Can be a [Range](#) object, an array of ranges, or a text constant that represents the name of another report. For an external database, **SourceData** is an array of strings containing the SQL query string, where each element is up to 255 characters in length. You should use the **Connection** argument to specify the ODBC connection string. For compatibility with earlier versions of Excel, **SourceData** can be a two-element array. The first element is the connection string specifying the ODBC source for the data. The second element is the SQL query string used to get the data. If you specify **SourceData**, you must also specify **SourceType**. If the active cell is inside the **SourceData** range, you must specify **TableDestination** as well.

**TableDestination** Optional **Variant**. A **Range** object specifying where the report should be placed on the worksheet. If this argument is omitted, the report is placed at the active cell.

**TableName** Optional **Variant**. A string that specifies the name of the new report.

**RowGrand** Optional **Variant**. **True** to show grand totals for rows in the report.

**ColumnGrand** Optional **Variant**. **True** to show grand totals for columns in the report.

**SaveData** Optional **Variant**. **True** to save data with the report. **False** to save only the report definition.

**HasAutoFormat** Optional **Variant**. **True** to have Microsoft Excel automatically format the report when it's refreshed or when fields are moved.

**AutoPage** Optional **Variant**. Valid only if **SourceType** is **xlConsolidation**.

**True** to have Microsoft Excel create a page field for the consolidation. If **AutoPage** is **False**, you must create the page field or fields.

**Reserved** Optional **Variant**. Not used by Microsoft Excel.

**BackgroundQuery** Optional **Variant**. **True** to have Excel perform queries for the report asynchronously (in the background). The default value is **False**.

**OptimizeCache** Optional **Variant**. **True** to optimize the PivotTable cache when it's constructed. The default value is **False**.

**PageFieldOrder** Optional **Variant**. The order in which page fields are added to the PivotTable report's layout. Can be one of the following **XIOrder** constants: **xlDownThenOver** or **xlOverThenDown**. The default value is **xlDownThenOver**.

**PageFieldWrapCount** Optional **Variant**. The number of page fields in each column or row in the PivotTable report. The default value is 0 (zero).

**ReadData** Optional **Variant**. **True** to create a PivotTable cache that contains all records from the external database; this cache can be very large. If **ReadData** is **False**, you can set some of the fields as server-based page fields before the data is actually read.

**Connection** Optional **Variant**. A string that contains ODBC settings that allow Excel to connect to an ODBC data source. The connection string has the form "ODBC;<connection string>". This argument overrides any previous setting for the [PivotCache](#) object's [Connection](#) property.

## Example

This example creates a new PivotTable report from a Microsoft Excel database (contained in the range A1:C100).

```
ActiveSheet.PivotTableWizard xlDatabase, Range("A1:C100")
```



# Play Method

This method should not be used. Sound notes have been removed from Microsoft Excel.



# Points Method

Returns an object that represents a single point (a [Point](#) object) or a collection of all the points (a [Points](#) collection) in the series. Read-only.

*expression*.**Points**(*Index*)

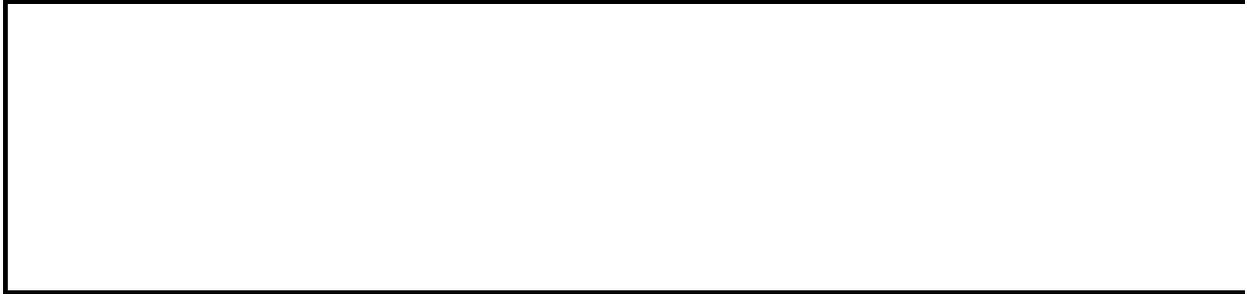
*expression* Required. An expression that returns a **Series** object.

*Index* Optional **Variant**. The name or number of the point.

## Example

This example applies a data label to point one in series one in Chart1.

```
Charts("Chart1").SeriesCollection(1).Points(1).ApplyDataLabels
```



# PointsToScreenPixelsX Method

Converts a horizontal measurement from points (document coordinates) to screen pixels (screen coordinates). Returns the converted measurement as a **Long** value.

*expression*.**PointsToScreenPixelsX**(*Points*)

*expression* An expression that returns a **Window** object.

**Points** Required **Long**. The number of points horizontally along the top of the document window, starting from the left.

## Example

This example determines the height and width (in pixels) of the selected cells in the active window and returns the values in the `lwinWidth` and `lwinHeight` variables.

```
With Activewindow
    lwinWidth = _
        .PointsToScreenPixelsX(.Selection.Width)
    lwinHeight = _
        .PointsToScreenPixelsY(.Selection.Height)
End With
```



# PointsToScreenPixelsY Method

Converts a vertical measurement from points (document coordinates) to screen pixels (screen coordinates). Returns the converted measurement as a **Long** value.

*expression*.**PointsToScreenPixelsY**(*Points*)

*expression* An expression that returns a **Window** object.

**Points** Required **Long**. The number of points vertically along the left edge of the document window, starting from the top.

## Example

This example determines the height and width (in pixels) of the selected cells in the active window and returns the values in the `lwinWidth` and `lwinHeight` variables.

```
With ActiveWindow
    lwinWidth = _
        .PointsToScreenPixelsX(.Selection.Width)
    lwinHeight = _
        .PointsToScreenPixelsY(.Selection.Height)
End With
```



# Post Method

Posts the specified workbook to a public folder. This method works only with a Microsoft Exchange client connected to a Microsoft Exchange server.

*expression*.**Post**(*DestName*)

*expression* Required. An expression that returns a **Workbook** object.

*DestName* Optional **Variant**. This argument is ignored. The **Post** method prompts the user to specify the destination for the workbook.

## Example

This example posts the active workbook.

ActiveWorkbook.**Post**



[Show All](#)

# PresetDrop Method

Specifies whether the callout line attaches to the top, bottom, or center of the callout text box or whether it attaches at a point that's a specified distance from the top or bottom of the text box.

*expression*.PresetDrop(*DropType*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**DropType** Required [MsoCalloutDropType](#). The starting position of the callout line relative to the text bounding box

MsoCalloutDropType can be one of these MsoCalloutDropType constants.

**msoCalloutDropBottom**

**msoCalloutDropCenter**

**msoCalloutDropCustom** Specifying msoCalloutDropCustom for this argument will cause your code to fail.

**msoCalloutDropMixed**

**msoCalloutDropTop**

## Example

This example specifies that the callout line attach to the top of the text bounding box for shape one on myDocument. For the example to work, shape one must be a callout.

```
Set myDocument = Worksheets(1)
myDocument.Shapes(1).Callout.PresetDrop msoCalloutDropTop
```

This example toggles between two preset drops for shape one on myDocument. For the example to work, shape one must be a callout.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(1).Callout
    If .DropType = msoCalloutDropTop Then
        .PresetDrop msoCalloutDropBottom
    ElseIf .DropType = msoCalloutDropBottom Then
        .PresetDrop msoCalloutDropTop
    End If
End With
```



[Show All](#)

# PresetGradient Method

 [PresetGradient method as it applies to the FillFormat object.](#)

Sets the specified fill to a preset gradient.

*expression*.**PresetGradient**(*Style*, *Variant*, *PresetGradientType*)

*expression* Required. An expression that returns one of the above objects.

*Style* Required [MsoGradientStyle](#).

MsoGradientStyle can be one of these MsoGradientStyle constants.

**msoGradientDiagonalDown**

**msoGradientDiagonalUp**

**msoGradientFromCenter**

**msoGradientFromCorner**

**msoGradientFromTitle**

**msoGradientHorizontal**

**msoGradientMixed**

**msoGradientVertical**

*Variant* Required **Integer**. The gradient variant. Can be a value from 1 through 4, corresponding to one of the four variants on the **Gradient** tab in the **Fill Effects** dialog box. If *GradientStyle* is **msoGradientFromCenter**, the *Variant* argument can only be 1 or 2.

*PresetGradientType* Required [MsoPresetGradientType](#).

MsoPresetGradientType can be one of these MsoPresetGradientType constants.

**msoGradientRainbow**

**msoGradientBrass**

**msoGradientCalmWater**

**msoGradientChrome**

**msoGradientChromeII**  
**msoGradientDaybreak**  
**msoGradientDesert**  
**msoGradientEarlySunset**  
**msoGradientFire**  
**msoGradientFog**  
**msoGradientGold**  
**msoGradientGoldII**  
**msoGradientHorizon**  
**msoGradientLateSunset**  
**msoGradientMahogany**  
**msoGradientMoss**  
**msoGradientNightfall**  
**msoGradientOcean**  
**msoGradientParchment**  
**msoGradientPeacock**  
**msoGradientRainbowII**  
**msoGradientSapphire**  
**msoGradientSilver**  
**msoGradientWheat**  
**msoPresetGradientMixed**

 [PresetGradient](#) method as it applies to the **ChartFillFormat** object.

Sets the specified fill to a preset gradient.

*expression*.**PresetGradient**(*Style*, *Variant*, *PresetGradientType*)

*expression* Required. An expression that returns one of the above objects.

*Style* Required [MsoGradientStyle](#).

MsoGradientStyle can be one of these MsoGradientStyle constants.

**msoGradientDiagonalDown**  
**msoGradientDiagonalUp**

**msoGradientFromCenter**  
**msoGradientFromCorner**  
**msoGradientFromTitle**  
**msoGradientHorizontal**  
**msoGradientMixed**  
**msoGradientVertical**

**Variant** Required **Long**. The gradient variant. Can be a value from 1 through 4, corresponding to one of the four variants on the **Gradient** tab in the **Fill Effects** dialog box. If **GradientStyle** is **msoGradientFromCenter**, the **Variant** argument can only be 1 or 2.

**PresetGradientType** Required [MsoPresetGradientType](#).

MsoPresetGradientType can be one of these MsoPresetGradientType constants.

**msoGradientRainbow**  
**msoGradientBrass**  
**msoGradientCalmWater**  
**msoGradientChrome**  
**msoGradientChromeII**  
**msoGradientDaybreak**  
**msoGradientDesert**  
**msoGradientEarlySunset**  
**msoGradientFire**  
**msoGradientFog**  
**msoGradientGold**  
**msoGradientGoldII**  
**msoGradientHorizon**  
**msoGradientLateSunset**  
**msoGradientMahogany**  
**msoGradientMoss**  
**msoGradientNightfall**  
**msoGradientOcean**  
**msoGradientParchment**

**msoGradientPeacock**  
**msoGradientRainbowII**  
**msoGradientSapphire**  
**msoGradientSilver**  
**msoGradientWheat**  
**msoPresetGradientMixed**

## Example

This example sets the fill format for chart two to the same style used for chart one.

```
Set c1f = Charts(1).ChartArea.Fill
If c1f.Type = msoFillGradient Then
    With Charts(2).ChartArea.Fill
        .Visible = True
        .PresetGradient c1f.GradientStyle, _
            c1f.GradientVariant, c1f.PresetGradientType
    End With
End If
```



[Show All](#)

# PresetTextured Method

Sets the specified fill format to a preset texture.

*expression*.PresetTextured(*PresetTexture*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*PresetTexture* Required [MsoPresetTexture](#).

MsoPresetTexture can be one of these MsoPresetTexture constants.

**msoPresetTextureMixed**

**msoTextureBlueTissuePaper**

**msoTextureBouquet**

**msoTextureBrownMarble**

**msoTextureCanvas**

**msoTextureCork**

**msoTextureDenim**

**msoTextureFishFossil**

**msoTextureGranite**

**msoTextureGreenMarble**

**msoTextureMediumWood**

**msoTextureNewsprint**

**msoTextureOak**

**msoTexturePaperBag**

**msoTexturePapyrus**

**msoTextureParchment**

**msoTexturePinkTissuePaper**

**msoTexturePurpleMesh**

**msoTextureRecycledPaper**

**msoTextureSand**

**msoTextureStationery**

**msoTextureWalnut**

**msoTextureWaterDroplets**

**msoTextureWhiteMarble**

**msoTextureWovenMat**

## Example

This example sets the fill format for chart two to the same style used for chart one.

```
Set c1f = Charts(1).ChartArea.Fill
If c1f.Type = msoFillTextured Then
    With Charts(2).ChartArea.Fill
        .Visible = True
        If c1f.TextureType = msoTexturePreset Then
            .PresetTextured c1f.PresetTexture
        Else
            .UserTextured c1f.TextureName
        End If
    End With
End If
```



# Previous Method

Returns a **Comment** object that represents the previous comment.

*expression*.**Previous**

*expression* Required. An expression that returns a **Comment** object.

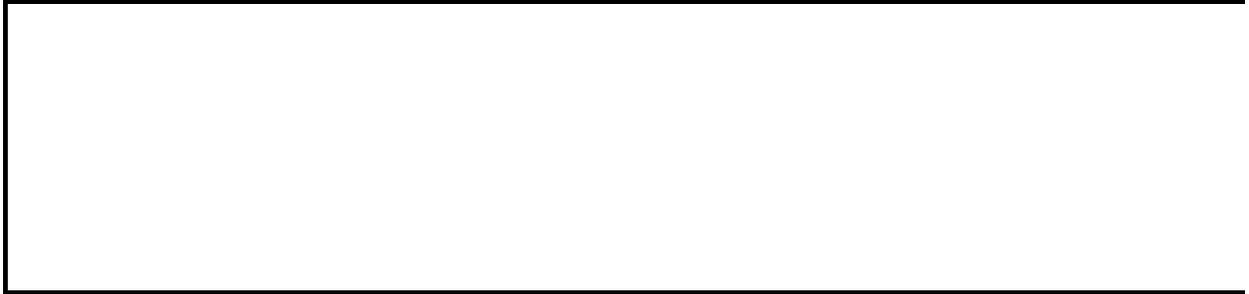
## Remarks

This method works only on one sheet. Using this method on the first comment on a sheet returns **Null** (not the last comment on the previous sheet).

## Example

This example hides the previous comment.

```
Range("a1").Comment.Previous.Visible = False
```



# PrevNode Method

Returns a **DiagramNode** object that represents the previous diagram node in a collection of diagram nodes.

*expression*.**PrevNode**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example adds additional child nodes to the first child node in a newly-created diagram.

```
Sub AddToPrevNode()  
  
    Dim dgnRoot As DiagramNode  
    Dim shpDiagram As Shape  
    Dim dgnPrev As DiagramNode  
    Dim intCount As Integer  
  
    'Add organizational chart to the current document  
    Set shpDiagram = ActiveSheet.Shapes.AddDiagram( _  
        Type:=msoDiagramOrgChart, _  
        Left:=10, _  
        Top:=15, _  
        Width:=400, _  
        Height:=475)  
  
    'Add first diagram node  
    Set dgnRoot = shpDiagram.DiagramNode.Children.AddNode  
  
    'Add three child nodes off the first diagram node  
    For intCount = 1 To 3  
        dgnRoot.Children.AddNode  
    Next intCount  
  
    'Access the node immediately preceding  
    'the second diagram node  
    Set dgnPrev = dgnRoot.Children.Item(2).PrevNode  
  
    'Add three child nodes to the node immediately  
    'preceding the second node  
    For intCount = 1 To 3  
        dgnPrev.Children.AddNode  
    Next intCount  
  
End Sub
```



# PrintOut Method

Prints the object.

*expression*.**PrintOut**(*From, To, Copies, Preview, ActivePrinter, PrintToFile, Collate, PrToFileName*)

*expression* Required. An expression that returns an object in the Applies To list.

**From** Optional **Variant**. The number of the page at which to start printing. If this argument is omitted, printing starts at the beginning.

**To** Optional **Variant**. The number of the last page to print. If this argument is omitted, printing ends with the last page.

**Copies** Optional **Variant**. The number of copies to print. If this argument is omitted, one copy is printed.

**Preview** Optional **Variant**. **True** to have Microsoft Excel invoke print preview before printing the object. **False** (or omitted) to print the object immediately.

**ActivePrinter** Optional **Variant**. Sets the name of the active printer.

**PrintToFile** Optional **Variant**. **True** to print to a file. If **PrToFileName** is not specified, Microsoft Excel prompts the user to enter the name of the output file.

**Collate** Optional **Variant**. **True** to collate multiple copies.

**PrToFileName** Optional **Variant**. If **PrintToFile** is set to **True**, this argument specifies the name of the file you want to print to.

## Remarks

"Pages" in the descriptions of ***From*** and ***To*** refers to printed pages— not overall pages in the sheet or workbook.

## Example

This example prints the active sheet.

```
ActiveSheet.PrintOut
```



# PrintPreview Method

Shows a preview of the object as it would look when printed.

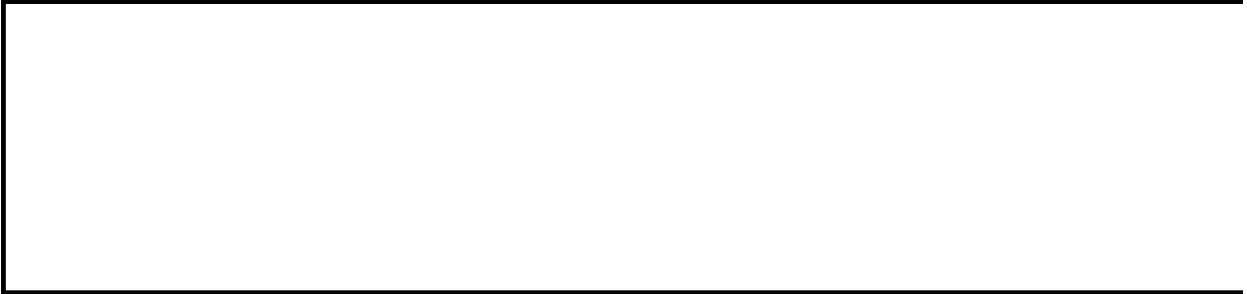
*expression*.**PrintPreview**

*expression* Required. An expression that returns an object in the Applies To list.

## Example

This example displays Sheet1 in print preview.

```
worksheets("Sheet1").PrintPreview
```



[Show All](#)

# Protect Method

 [Protect method as it applies to the \*\*Chart\*\* object.](#)

Protects a chart so that it cannot be modified.

*expression*.**Protect**(*Password*, *DrawingObjects*, *Contents*, *Scenarios*, *UserInterfaceOnly*)

*expression* Required. An expression that returns a **Chart** object.

**Password** Optional **Variant**. A string that specifies a case-sensitive password for the worksheet or workbook. If this argument is omitted, you can unprotect the worksheet or workbook without using a password. Otherwise, you must specify the password to unprotect the worksheet or workbook. If you forget the password, you cannot unprotect the worksheet or workbook. It's a good idea to keep a list of your passwords and their corresponding document names in a safe place.

**Note** Use strong passwords that combine upper- and lowercase letters, numbers, and symbols. Weak passwords don't mix these elements. Strong password: Y6dh!et5. Weak password: House27. Use a strong password that you can remember so that you don't have to write it down.

**DrawingObjects** Optional **Variant**. **True** to protect shapes. The default value is **False**.

**Contents** Optional **Variant**. **True** to protect contents. For a chart, this protects the entire chart. For a worksheet, this protects the locked cells. The default value is **True**.

**Scenarios** Optional **Variant**. **True** to protect scenarios. This argument is valid only for worksheets. The default value is **True**.

**UserInterfaceOnly** Optional **Variant**. **True** to protect the user interface, but not macros. If this argument is omitted, protection applies both to macros and to the user interface.

 [Protect method as it applies to the \*\*Worksheet\*\* object.](#)

Protects a worksheet so that it cannot be modified.

*expression*.**Protect**(*Password*, *DrawingObjects*, *Contents*, *Scenarios*, *UserInterfaceOnly*, *AllowFormattingCells*, *AllowFormattingColumns*, *AllowFormattingRows*, *AllowInsertingColumns*, *AllowInsertingRows*, *AllowInsertingHyperlinks*, *AllowDeletingColumns*, *AllowDeletingRows*, *AllowSorting*, *AllowFiltering*, *AllowUsingPivotTables*)

*expression* Required. An expression that returns a **Worksheet** object.

**Password** Optional **Variant**. A string that specifies a case-sensitive password for the worksheet or workbook. If this argument is omitted, you can unprotect the worksheet or workbook without using a password. Otherwise, you must specify the password to unprotect the worksheet or workbook. If you forget the password, you cannot unprotect the worksheet or workbook. It's a good idea to keep a list of your passwords and their corresponding document names in a safe place.

**DrawingObjects** Optional **Variant**. **True** to protect shapes. The default value is **False**.

**Contents** Optional **Variant**. **True** to protect contents. For a chart, this protects the entire chart. For a worksheet, this protects the locked cells. The default value is **True**.

**Scenarios** Optional **Variant**. **True** to protect scenarios. This argument is valid only for worksheets. The default value is **True**.

**UserInterfaceOnly** Optional **Variant**. **True** to protect the user interface, but not macros. If this argument is omitted, protection applies both to macros and to the user interface.

**AllowFormattingCells** Optional **Variant**. **True** allows the user to format any cell on a protected worksheet. The default value is **False**.

**AllowFormattingColumns** Optional **Variant**. **True** allows the user to format any column on a protected worksheet. The default value is **False**.

***AllowFormattingRows*** Optional **Variant**. **True** allows the user to format any row on a protected. The default value is **False**.

***AllowInsertingColumns*** Optional **Variant**. **True** allows the user to insert columns on the protected worksheet. The default value is **False**.

***AllowInsertingRows*** Optional **Variant**. **True** allows the user to insert rows on the protected worksheet. The default value is **False**.

***AllowInsertingHyperlinks*** Optional **Variant**. **True** allows the user to insert hyperlinks on the worksheet. The default value is **False**.

***AllowDeletingColumns*** Optional **Variant**. **True** allows the user to delete columns on the protected worksheet, where every cell in the column to be deleted is unlocked. The default value is **False**.

***AllowDeletingRows*** Optional **Variant**. **True** allows the user to delete rows on the protected worksheet, where every cell in the row to be deleted is unlocked. The default value is **False**.

***AllowSorting*** Optional **Variant**. **True** allows the user to sort on the protected worksheet. Every cell in the sort range must be unlocked or unprotected. The default value is **False**.

***AllowFiltering*** Optional **Variant**. **True** allows the user to set filters on the protected worksheet. Users can change filter criteria but can not enable or disable an auto filter. Users can set filters on an existing auto filter. The default value is **False**.

***AllowUsingPivotTables*** Optional **Variant**. **True** allows the user to use pivot table reports on the protected worksheet. The default value is **False**.

## Remarks

If you apply the **Protect** method with the *UserInterfaceOnly* argument set to **True** to a worksheet and then save the workbook, the entire worksheet (not just the interface) will be fully protected when you reopen the workbook. To re-enable the user interface protection after the workbook is opened, you must again apply the **Protect** method with *UserInterfaceOnly* set to **True**.

If changes wanted to be made to a protected worksheet, it is possible to use the **Protect** method on a protected worksheet if the password is supplied. Also, another method would be to unprotect the worksheet, make the necessary changes, and then protect the worksheet again.

**Note** 'Unprotected' means the cell may be locked (Format Cells dialog) but is included in a range defined in the Allow Users to Edit Ranges dialog, and the user has unprotected the range with a password or been validated via NT permissions.



[Protect method as it applies to the \*\*Workbook\*\* object.](#)

Protects a workbook so that it cannot be modified.

*expression*.**Protect**(*Password*, *Structure*, *Windows*)

*expression* Required. An expression that returns a **Workbook** object.

**Password** Optional **VARIANT**. A string that specifies a case-sensitive password for the worksheet or workbook. If this argument is omitted, you can unprotect the worksheet or workbook without using a password. Otherwise, you must specify the password to unprotect the worksheet or workbook. If you forget the password, you cannot unprotect the worksheet or workbook. It's a good idea to keep a list of your passwords and their corresponding document names in a safe place.

**Structure** Optional **VARIANT**. **True** to protect the structure of the workbook (the relative position of the sheets). The default value is **False**.

**Windows** Optional **VARIANT**. **True** to protect the workbook windows. If this

argument is omitted, the windows aren't protected.

## Example

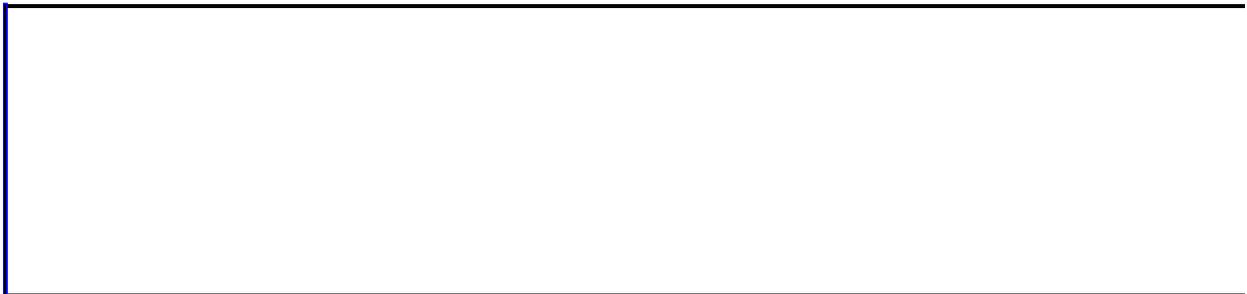
[As it applies to the \*\*Chart\*\* and \*\*Worksheet\*\* objects.](#)

This example protects the active worksheet. You can verify the worksheet is protected, by attempting to enter a value into any cell, on the active worksheet.

```
Sub ProtectSheet()  
    ActiveSheet.Protect Scenarios:=True, UserInterfaceOnly:=True  
End Sub
```

This example protects the active chart. You can verify the chart is protected, by attempting to enter a value into any cell, on the active worksheet. This example assumes a chart exists in the application.

```
Sub ProtectChart()  
    ActiveChart.Protect Scenarios:=True, UserInterfaceOnly:=True  
End Sub
```



# ProtectSharing Method

Saves the workbook and protects it for sharing.

*expression*.**ProtectSharing**(*Filename*, *Password*, *WriteResPassword*, *ReadOnlyRecommended*, *CreateBackup*, *SharingPassword*)

*expression* An expression that returns a **Workbook** object.

**Filename** Optional **Variant**. A string indicating the name of the saved file. You can include a full path; if you don't, Microsoft Excel saves the file in the current folder.

**Password** Optional **Variant**. A case-sensitive string indicating the protection password to be given to the file. Should be no longer than 15 characters.

**WriteResPassword** Optional **Variant**. A string indicating the write-reservation password for this file. If a file is saved with the password and the password isn't supplied when the file is opened, the file is opened read-only.

**ReadOnlyRecommended** Optional **Variant**. **True** to display a message when the file is opened, recommending that the file be opened read-only.

**CreateBackup** Optional **Variant**. **True** to create a backup file.

**SharingPassword** Optional **Variant**. A string indicating the password to be used to protect the file for sharing.

**Note** Use strong passwords that combine upper- and lowercase letters, numbers, and symbols. Weak passwords don't mix these elements. Strong password: Y6dh!et5. Weak password: House27. Use a strong password that you can remember so that you don't have to write it down.

## Example

This example saves workbook one and protects it for sharing.

```
Sub ProtectWorkbook()  
  
    Dim wksOne As Worksheet  
    Dim strPwd As String  
    Dim strSharePwd As String  
  
    Set wksOne = Application.ActiveSheet  
  
    strPwd = InputBox("Enter password for the file")  
    strSharePwd = InputBox("Enter password for sharing")  
  
    wksOne.ProtectSharing Password:=strPwd, _  
        SharingPassword:=strSharePwd  
  
End Sub
```



[Show All](#)

# Publish Method

 [Publish method as it applies to the \*\*PublishObject\*\* object or the \*\*PublishObjects\*\* collection.](#)

Saves an item or a collection of items in a document to a Web page.

*expression*.**Publish**(*Create*)

*expression* An expression that returns a [PublishObject](#) object or a [PublishObjects](#) collection.

**Create** Optional **Variant**. This argument is used only with a **PublishObject** object. If the HTML file exists, setting this argument to **True** replaces the file, and setting this argument to **False** inserts the item or items at the end of the file. If the file does not exist, then the file is created regardless of the value of the **Create** argument.

## Remarks

The [FileName](#) property returns or sets the location and name of the HTML file.



[Publish method as it applies to the ListObject object.](#)

Publishes the **ListObject** object to a server that is running Microsoft Windows SharePoint Services. Returns a **String** which is the URL of the published list on the SharePoint site.

*expression.Publish(Target, LinkSource)*

*expression* Required. An expression that returns a **ListObject** object.

**Target** Required **Variant**. Contains an **Array** of **Strings**. The following table describes the elements of this array:

<b>Element#</b>	<b>Contents</b>
0	URL of SharePoint server
1	ListName (Display Name)
2	Description of the list. Optional.

**LinkSource** Required **Boolean**.

## Remarks

If the **ListObject** object is not currently linked to a list on a SharePoint site, setting **LinkSource** to **True** will create a new list on the specified SharePoint site. If the **ListObject** object is currently linked to a SharePoint site, setting **LinkSource** argument to **True** will replace the existing link (you can only link the list to one SharePoint site). If the **ListObject** object is not currently linked, setting **LinkSource** to **False** will leave the **ListObject** object unlinked. If the **ListObject** object is currently linked to a SharePoint site, setting **LinkSource** to **False** will keep the **ListObject** object linked to the current SharePoint site.

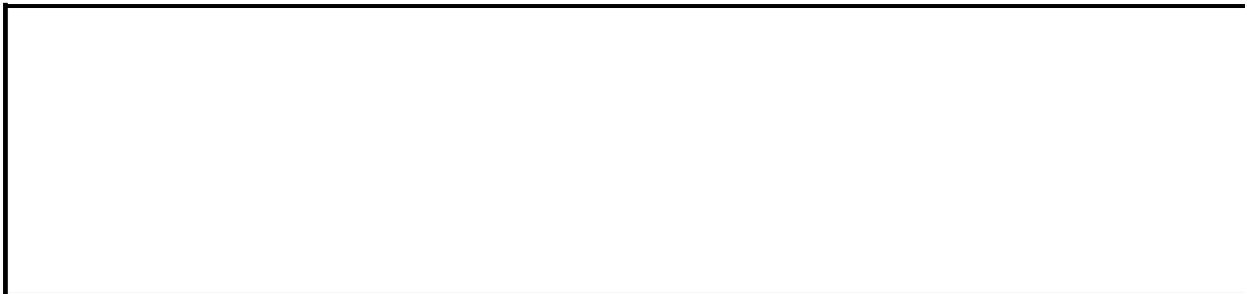
## Example

This example saves the range D5:D9 on the First Quarter worksheet in the active workbook to a Web page named stockreport.htm. The Spreadsheet component is used to make the Web page interactive.

```
ActiveWorkbook.PublishObjects.Add( _  
    SourceType:=xlSourceRange, _  
    Filename:"\\Server2\Q1\stockreport.htm", _  
    Sheet:"First Quarter", _  
    Source:"D5:D9", _  
    HTMLType:=xlHTMLCalc).Publish
```

The following example sets elements of the *Target* parameter to publish the **ListObject** object to the SharePoint site named "MyServer". In this example, it is assumed that the **ListObject** object is not currently linked to another list source, so setting the *LinkSource* argument to **True** creates a new two-way link between this list and the list on the target SharePoint site.

```
Dim objWrksht as Worksheet  
Dim objListobj as ListObject  
Dim strURL as String  
  
Set objWrksht = ActiveWorkbook.Worksheets("Sheet1")  
Set objListObj = objWrksht.ListObjects(1)  
  
strURL = objListObj.Publish(Array("HTTP://MyServer", "MyList", "Desc
```



# PurgeChangeHistoryNow Method

Removes entries from the change log for the specified workbook.

*expression*.**PurgeChangeHistoryNow**(*Days*, *SharingPassword*)

*expression* An expression that returns a **Workbook** object.

**Days** Required **Long**. The number of days that changes in the change log are to be retained.

**SharingPassword** Optional **Variant**. The password that unprotects the workbook for sharing. If the workbook is protected for sharing with a password and this argument is omitted, the user is prompted for the password.

## Example

This example removes all changes that are more than one day old from the change log for the active workbook.

```
ActiveWorkbook.PurgeChangeHistoryNow Days:=1
```



# Quit Method

Quits Microsoft Excel.

*expression*.**Quit**

*expression* Required. An expression that returns an **Application** object.

## Remarks

If unsaved workbooks are open when you use this method, Microsoft Excel displays a dialog box asking whether you want to save the changes. You can prevent this by saving all workbooks before using the **Quit** method or by setting the [DisplayAlerts](#) property to **False**. When this property is **False**, Microsoft Excel doesn't display the dialog box when you quit with unsaved workbooks; it quits without saving them.

If you set the [Saved](#) property for a workbook to **True** without saving the workbook to the disk, Microsoft Excel will quit without asking you to save the workbook.

## Example

This example saves all open workbooks and then quits Microsoft Excel.

```
For Each w In Application.Workbooks  
    w.Save  
Next w  
Application.Quit
```



# RadarGroups Method

On a 2-D chart, returns an object that represents either a single radar chart group (a [ChartGroup](#) object) or a collection of the radar chart groups (a [ChartGroups](#) collection).

*expression*.**RadarGroups**(*Index*)

*expression* Required. An expression that returns a **Chart** object.

*Index* Optional **Variant**. Specifies the chart group.

## Example

This example sets radar group one in Chart1 to use a different color for each data marker. The example should be run on a 2-D chart.

```
Charts("Chart1").RadarGroups(1).VaryByCategories = True
```



# RangeFromPoint Method

Returns the [Shape](#) or [Range](#) object that is positioned at the specified pair of screen coordinates. If there isn't a shape located at the specified coordinates, this method returns **Nothing**.

*expression*.**RangeFromPoint**(*x*, *y*)

*expression* An expression that returns a **Window** object.

**x** Required **Long**. The value (in pixels) that represents the horizontal distance from the left edge of the screen, starting at the top.

**y** Required **Long**. The value (in pixels) that represents the vertical distance from the top of the screen, starting on the left.

## Example

This example returns the alternative text for the shape immediately below the mouse pointer if the shape is a chart, line, or picture.

```
Private Function AltText(ByVal intMouseX As Integer, _
    ByVal intMouseY as Integer) As String
    Set objShape = ActiveWindow.RangeFromPoint _
        (x:=intMouseX, y:=intMouseY)
    If Not objShape Is Nothing Then
        With objShape
            Select Case .Type
                Case msoChart, msoLine, msoPicture:
                    AltText = .AlternativeText
                Case Else:
                    AltText = ""
            End Select
        End With
    Else
        AltText = ""
    End If
End Function
```



# RecheckSmartTags Method

Causes a foreground smart tag check to occur automatically annotating data that was not annotated before.

*expression*.**RecheckSmartTags**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example rechecks smart tags on the active workbook in the foreground.

```
Sub UseRecheckSmartTags()  
    ActiveWorkbook.RecheckSmartTags  
End Sub
```



# Record Method

This method should not be used. Sound notes have been removed from Microsoft Excel.



# RecordMacro Method

Records code if the macro recorder is on.

*expression*.**RecordMacro**(*BasicCode*, *XlmCode*)

*expression* Required. An expression that returns an **Application** object.

**BasicCode** Optional **Variant**. A string that specifies the Visual Basic code that will be recorded if the macro recorder is recording into a Visual Basic module. The string will be recorded on one line. If the string contains a carriage return (ASCII character 10, or Chr\$(10) in code), it will be recorded on more than one line.

**XlmCode** Optional **Variant**. This argument is ignored.

## Remarks

The **RecordMacro** method cannot record into the active module (the module in which the **RecordMacro** method exists).

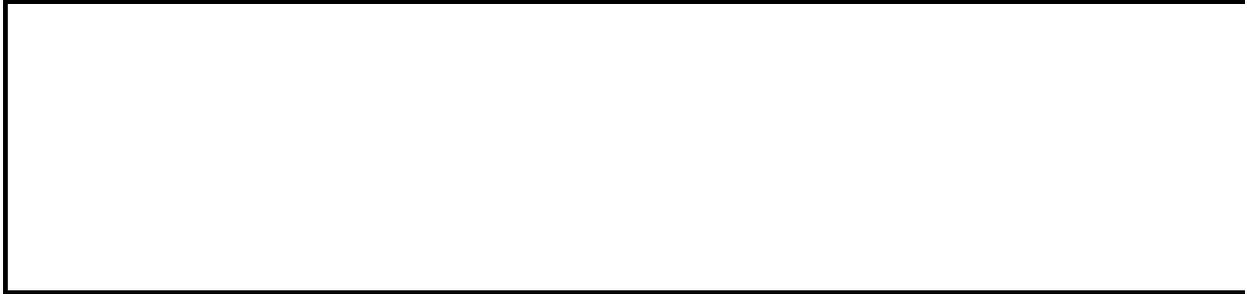
If **BasicCode** is omitted and the application is recording into Visual Basic, Microsoft Excel will record a suitable `Application.Run` statement.

To prevent recording (for example, if the user cancels your dialog box), call this function with two empty strings.

## Example

This example records Visual Basic code.

```
Application.RecordMacro BasicCode:="Application.Run ""MySub"" "
```



[Show All](#)

# Refresh Method

 [Refresh method as it applies to the \*\*ListObject\*\* object.](#)

Retrieves the current data and schema for the list from the server that is running Microsoft Windows SharePoint Services. This method can be used only with lists that are linked to a SharePoint site. If the SharePoint site is not available, calling this method will return an error.

*expression*.**Refresh()**

*expression* Required. An expression that returns a **ListObject** object.

## Remarks

Calling the **Refresh** method does not commit changes to the list in the Excel workbook. Uncommitted changes in the list in Excel are discarded when the **Refresh** method is called. To avoid losing any uncommitted changes, call the [UpdateChanges](#) method of the **ListObject** object before calling the **Refresh** method.



[Refresh method as it applies to the QueryTable object.](#)

Updates an external data range ([QueryTable](#)). **Boolean**.

*expression.Refresh(BackgroundQuery)*

*expression* Required. An expression that returns a [QueryTable](#) object.

**BackgroundQuery** Optional **Variant**. Used only with **QueryTables** that are based on the results of a SQL query. **True** to return control to the procedure as soon as a database connection is made and the the query is submitted. The **QueryTable** is updated in the background. **False** to return control to the procedure only after all data has been fetched to the worksheet. If this argument isn't specified, the setting of the [BackgroundQuery](#) property determines the query mode.

## Remarks

The following remarks apply to **QueryTable** objects that are based on the results of a SQL query.

The **Refresh** method causes Microsoft Excel to connect to the data source of the **QueryTable** object, execute the SQL query, and return data to the range that is based on the **QueryTable** object. Unless this method is called, the **QueryTable** object doesn't communicate with the data source.

When making the connection to the OLE DB or ODBC data source, Microsoft Excel uses the connection string specified by the **Connection** property. If the specified connection string is missing required values, dialog boxes will be displayed to prompt the user for the required information. If the **DisplayAlerts** property is **False**, dialog boxes aren't displayed and the **Refresh** method fails with the Insufficient Connection Information exception.

After Microsoft Excel makes a successful connection, it stores the completed connection string so that prompts won't be displayed for subsequent calls to the **Refresh** method during the same editing session. You can obtain the completed connection string by examining the value of the **Connection** property.

After the database connection is made, the SQL query is validated. If the query isn't valid, the **Refresh** method fails with the SQL Syntax Error exception.

If the query requires parameters, the **Parameters** collection must be initialized with parameter binding information before the **Refresh** method is called. If not enough parameters have been bound, the **Refresh** method fails with the Parameter Error exception. If parameters are set to prompt for their values, dialog boxes are displayed to the user regardless of the setting of the **DisplayAlerts** property. If the user cancels a parameter dialog box, the **Refresh** method halts and returns **False**. If extra parameters are bound with the **Parameters** collection, these extra parameters are ignored.

The **Refresh** method returns **True** if the query is successfully completed or started; it returns **False** if the user cancels a connection or parameter dialog box.

To see whether the number of fetched rows exceeded the number of available

rows on the worksheet, examine the [FetchedRowOverflow](#) property. This property is initialized every time the **Refresh** method is called.

[Refresh method as it applies to the \*\*Chart\*\* and \*\*PivotCache\*\* objects.](#)

Updates the cache of the [Chart](#) or [PivotTable](#) object.

*expression*.**Refresh**

*expression* Required. An expression that returns one of the above objects. For the **PivotCache** object, the cache must have at least one PivotTable report associated with it.

[Refresh method as it applies to the \*\*XmlDataBinding\*\* object.](#)

Retrieves XML data using the current connection settings of the specified [XmlDataBinding](#) object. Returns [IXmlImportResult](#).

IXmlImportResult can be one of the following IXmlImportResult constants  
**xlXmlImportElementsTruncated** The contents of the specified XML data file have been truncated because the XML data file is too large for the worksheet.

**xlXmlImportSuccess** The XML data file was successfully imported.

**xlXmlImportValidationFailed** The contents of the XML data file do not match the specified schema map.

*expression*.**Refresh()**

*expression* Required. An expression that returns an [XmlDataBinding](#) object.

## Example

This example refreshes the PivotTable cache for the first PivotTable report on the first worksheet in a workbook.

```
Worksheets(1).PivotTables(1).PivotCache.Refresh
```



# RefreshAll Method

Refreshes all external data ranges and PivotTable reports in the specified workbook.

*expression*.**RefreshAll**

*expression* Required. An expression that returns a **Workbook** object.

## Remarks

Objects that have the [BackgroundQuery](#) property set to **True** are refreshed in the background.

## Example

This example refreshes all external data ranges and PivotTable reports in the third workbook.

```
Workbooks(3).RefreshAll
```



[Show All](#)

# RefreshData Method

 [RefreshData method as it applies to the \*\*IRtdServer\*\* object.](#)

This method is called by Microsoft Excel to get new data. Returns a **Variant**.

*expression.RefreshData(TopicCount)*

*expression* Required. An expression that returns an [IRtdServer](#) object.

**TopicCount** Required **Long**. The RTD server must change the value of the **TopicCount** to the number of elements in the array returned.

## Remarks

The data returned to Excel is a **Variant** containing a two-dimensional array. The first dimension represents the list of topic IDs. The second dimension represents the values associated with the topic IDs.

 [RefreshData method as it applies to the \*\*RTD\*\* object.](#)

Requests an update of real-time data from the real-time data server.

*expression*.**RefreshData()**

*expression* Required. An expression that returns an [RTD](#) object.

## Remarks

Avoid using the **RefreshData** method in user-defined functions because this method will fail if it is called during recalculation.

--

# RefreshTable Method

Refreshes the PivotTable report from the source data. Returns **True** if it's successful.

*expression*.**RefreshTable**

*expression* Required. An expression that returns a **PivotTable** object.

## Example

This example refreshes the PivotTable report.

```
Set pvtTable = Worksheets("Sheet1").Range("A3").PivotTable  
pvtTable.RefreshTable
```



# RegisterXLL Method

Loads an XLL code resource and automatically registers the functions and commands contained in the resource.

*expression*.**RegisterXLL**(*Filename*)

*expression* Required. An expression that returns an **Application** object.

*Filename* Required **String**. Specifies the name of the XLL to be loaded.

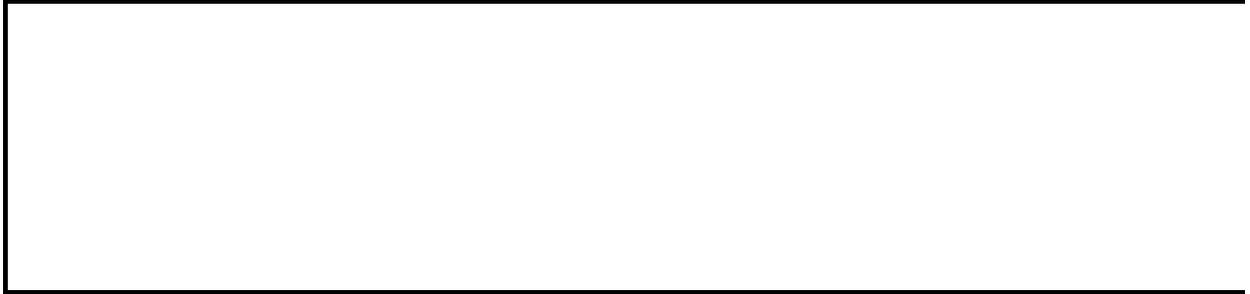
## Remarks

This method returns **True** if the code resource is successfully loaded; otherwise, the method returns **False**.

## Example

This example loads an XLL file and registers the functions and commands in the file.

```
Application.RegisterXLL "XLMAPI.XLL"
```



# Regroup Method

Regroups the group that the specified shape range belonged to previously. Returns the regrouped shapes as a single **Shape** object.

*expression*.**Regroup**

*expression* Required. An expression that returns a **ShapeRange** object.

## Remarks

The **Regroup** method only restores the group for the first previously grouped shape it finds in the specified **ShapeRange** collection. Therefore, if the specified shape range contains shapes that previously belonged to different groups, only one of the groups will be restored.

Note that because a group of shapes is treated as a single shape, grouping and ungrouping shapes changes the number of items in the **Shapes** collection and changes the index numbers of items that come after the affected items in the collection.

## Example

This example regroups the shapes in the selection in the active window. If the shapes haven't been previously grouped and ungrouped, this example will fail.

```
ActiveWindow.Selection.ShapeRange.Regroup
```



# RejectAllChanges Method

Rejects all changes in the specified shared workbook.

*expression*.**AcceptAllChanges**(*When*, *Who*, *Where*)

*expression* Required. An expression that returns a **Workbook** object

**When** Optional **Variant**. Specifies when all the changes are rejected.

**Who** Optional **Variant**. Specifies by whom all the changes are rejected.

**Where** Optional **Variant**. Specifies where all the changes are rejected.

## Example

This example rejects all changes in the active workbook.

`ActiveWorkbook.RejectAllChanges`



[Show All](#)

# ReloadAs Method

Reloads a workbook based on an HTML document, using the specified document encoding.

*expression*.**ReloadAs**(*Encoding*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*Encoding* Required [MsoEncoding](#). The encoding that is to be applied to the workbook.

MsoEncoding can be one of these MsoEncoding constants.

**msoEncodingEBCDICArabic**

**msoEncodingArabic**

**msoEncodingArabicASMO**

**msoEncodingArabicAutoDetect**

**msoEncodingArabicTransparentASMO**

**msoEncodingAutoDetect**

**msoEncodingBaltic**

**msoEncodingCentralEuropean**

**msoEncodingCyrillic**

**msoEncodingCyrillicAutoDetect**

**msoEncodingEBCDICDenmarkNorway**

**msoEncodingEBCDICFinlandSweden**

**msoEncodingEBCDICFrance**

**msoEncodingEBCDICGermany**

**msoEncodingEBCDIGreek**

**msoEncodingEBCDIGreekModern**

**msoEncodingEBCDICHebrew**

**msoEncodingEBCDICIcelandic**

**msoEncodingEBCDICInternational**

**msoEncodingEBCDICItaly**  
**msoEncodingEBCDICJapaneseKatakanaExtended**  
**msoEncodingEBCDICJapaneseKatakanaExtendedAndJapanese**  
**msoEncodingEBCDICJapaneseLatinExtendedAndJapanese**  
**msoEncodingEBCDIKKoreanExtended**  
**msoEncodingEBCDIKKoreanExtendedAndKorean**  
**msoEncodingEBCDICLatinAmericaSpain**  
**msoEncodingEBCDICMultilingualROECELatin2**  
**msoEncodingEBCDICRussian**  
**msoEncodingEBCDICSerbianBulgarian**  
**msoEncodingEBCDICSimplifiedChineseExtendedAndSimplifiedChinese**  
**msoEncodingEBCDICThai**  
**msoEncodingEBCDICTurkish**  
**msoEncodingEBCDICTurkishLatin5**  
**msoEncodingEBCDICUnitedKingdom**  
**msoEncodingEBCDICUSCanada**  
**msoEncodingEBCDICUSCanadaAndJapanese**  
**msoEncodingEBCDICUSCanadaAndTraditionalChinese**  
**msoEncodingEUCChineseSimplifiedChinese**  
**msoEncodingEUCJapanese**  
**msoEncodingEUCKorean**  
**msoEncodingEUCTaiwaneseTraditionalChinese**  
**msoEncodingEuropa3**  
**msoEncodingExtAlphaLowercase**  
**msoEncodingGreek**  
**msoEncodingGreekAutoDetect**  
**msoEncodingHebrew**  
**msoEncodingHZGBSimplifiedChinese**  
**msoEncodingIA5German**  
**msoEncodingIA5IRV**  
**msoEncodingIA5Norwegian**  
**msoEncodingIA5Swedish**  
**msoEncodingISO2022CNSimplifiedChinese**

**msoEncodingISO2022CNTraditionalChinese**  
**msoEncodingISO2022JPJISX02011989**  
**msoEncodingISO2022JPJISX02021984**  
**msoEncodingISO2022JPNoHalfwidthKatakana**  
**msoEncodingISO2022KR**  
**msoEncodingISO6937NonSpacingAccent**  
**msoEncodingISO885915Latin9**  
**msoEncodingISO88591Latin1**  
**msoEncodingISO88592CentralEurope**  
**msoEncodingISO88593Latin3**  
**msoEncodingISO88594Baltic**  
**msoEncodingISO88595Cyrillic**  
**msoEncodingISO88596Arabic**  
**msoEncodingISO88597Greek**  
**msoEncodingISO88598Hebrew**  
**msoEncodingISO88599Turkish**  
**msoEncodingJapaneseAutoDetect**  
**msoEncodingJapaneseShiftJIS**  
**msoEncodingKOI8R**  
**msoEncodingKOI8U**  
**msoEncodingKorean**  
**msoEncodingKoreanAutoDetect**  
**msoEncodingKoreanJohab**  
**msoEncodingMacArabic**  
**msoEncodingMacCroatia**  
**msoEncodingMacCyrillic**  
**msoEncodingMacGreek1**  
**msoEncodingMacHebrew**  
**msoEncodingMacIcelandic**  
**msoEncodingMacJapanese**  
**msoEncodingMacKorean**  
**msoEncodingMacLatin2**  
**msoEncodingMacRoman**

**msoEncodingMacRomania**  
**msoEncodingMacSimplifiedChineseGB2312**  
**msoEncodingMacTraditionalChineseBig5**  
**msoEncodingMacTurkish**  
**msoEncodingMacUkraine**  
**msoEncodingOEMArabic**  
**msoEncodingOEMBaltic**  
**msoEncodingOEMCanadianFrench**  
**msoEncodingOEMCyrillic**  
**msoEncodingOEMCyrillicII**  
**msoEncodingOEMGreek437G**  
**msoEncodingOEMHebrew**  
**msoEncodingOEMIcelandic**  
**msoEncodingOEMModernGreek**  
**msoEncodingOEMMultilingualLatinI**  
**msoEncodingOEMMultilingualLatinII**  
**msoEncodingOEMNordic**  
**msoEncodingOEMPortuguese**  
**msoEncodingOEMTurkish**  
**msoEncodingOEMUnitedStates**  
**msoEncodingSimplifiedChineseAutoDetect**  
**msoEncodingSimplifiedChineseGBK**  
**msoEncodingT61**  
**msoEncodingTaiwanCNS**  
**msoEncodingTaiwanEten**  
**msoEncodingTaiwanIBM5550**  
**msoEncodingTaiwanTCA**  
**msoEncodingTaiwanTeleText**  
**msoEncodingTaiwanWang**  
**msoEncodingThai**  
**msoEncodingTraditionalChineseAutoDetect**  
**msoEncodingTraditionalChineseBig5**  
**msoEncodingTurkish**

**msoEncodingUnicodeBigEndian**  
**msoEncodingUnicodeLittleEndian**  
**msoEncodingUSASCII**  
**msoEncodingUTF7**  
**msoEncodingUTF8**  
**msoEncodingVietnamese**  
**msoEncodingWestern**

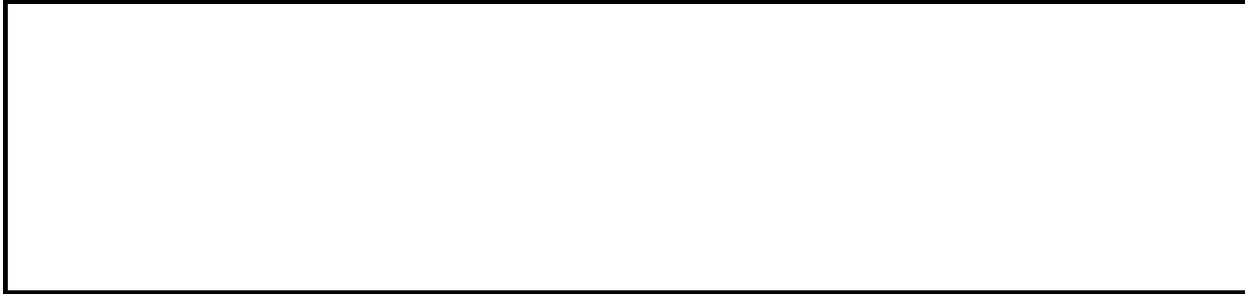
## Remarks

Only **msoEncoding** constants that are applicable to HTML work with the **ReloadAs** method.

## Example

This example reloads the first workbook, using Western document encoding.

```
Workbooks(1).ReloadAs Encoding:=msoEncodingWestern
```



# RemoveAllItems Method

Removes all entries from a Microsoft Excel list box or combo box. Use the [Clear](#) method to remove all items from an ActiveX list box or combo box.

*expression*.**RemoveAllItems**

*expression* Required. An expression that returns a **ControlFormat** object.

## Example

This example removes all items from a list box. If Shapes(2) doesn't represent a list box, this example fails.

```
Worksheets(1).Shapes(2).ControlFormat.RemoveAllItems
```



# RemoveItem Method

Removes one or more items from a list box or combo box.

*expression*.**RemoveItem**(*Index*, *Count*)

*expression* An expression that returns a **ControlFormat** object.

**Index** Required **Long**. The number of the first item to be removed. Valid values are from 1 to the number of items in the list (returned by the **ListCount** property).

**Count** Optional **Variant**. The number of items to be removed, starting at item **Index**. If this argument is omitted, one item is removed. If **Index + Count** exceeds the number of items in the list, all items from **Index** through the end of the list are removed without an error.

## Remarks

If the specified object has a fill range defined for it, this method fails.

Use the [RemoveAllItems](#) method to remove all entries from a Microsoft Excel list box or combo box. Use the [Clear](#) method to remove all items from an ActiveX list box or combo box.

## Example

This example removes the selected item from a list box. If Shapes(2) doesn't represent a list box, this example fails.

```
Set lbcf = Worksheets(1).Shapes(2).ControlFormat  
lbcf.RemoveItem lbcf.ListIndex
```



# RemoveSubtotal Method

Removes subtotals from a list.

*expression*.**RemoveSubtotal**

*expression* Required. An expression that returns a **Range** object.

## Example

This example removes subtotals from the range A1:G37 on Sheet1. The example should be run on a list that has subtotals.

```
Worksheets("Sheet1").Range("A1:G37").RemoveSubtotal
```



# RemoveUser Method

Disconnects the specified user from the shared workbook.

*expression*.**RemoveUser**(*Index*)

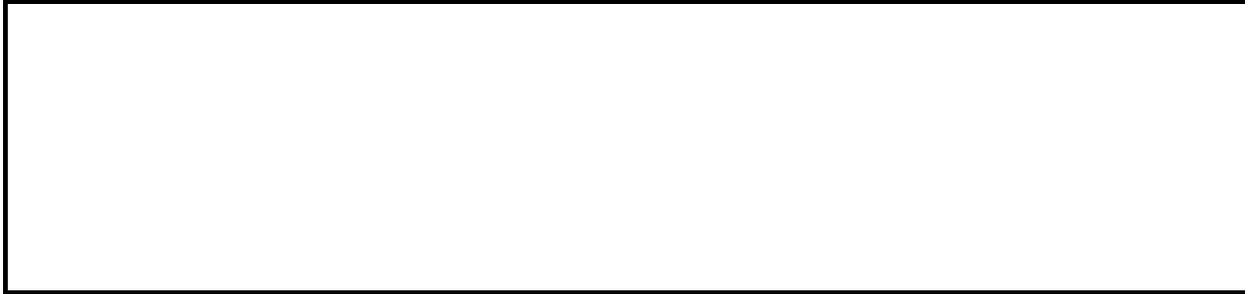
*expression* Required. An expression that returns a **Workbook** object.

*Index* Required **Long**. The user index.

## Example

This example disconnects user two from the shared workbook.

```
workbooks(2).RemoveUser 2
```



# Repeat Method

Repeats the last user-interface action.

*expression*.**Repeat**

*expression* Required. An expression that returns an **Application** object.

## **Remarks**

This method repeats only the last action taken by the user before running the macro, and it must be the first line in the macro. It cannot be used to repeat Visual Basic commands.

## Example

This example repeats the last user-interface command. The example must be the first line in a macro.

Application.**Repeat**



[Show All](#)

# Replace Method

 [Replace method as it applies to the \*\*Range\*\* object.](#)

Returns a **Boolean** indicating characters in cells within the specified range. Using this method doesn't change either the selection or the active cell.

*expression*.**Replace**(*What*, *Replacement*, *LookAt*, *SearchOrder*, *MatchCase*, *MatchByte*, *SearchFormat*, *ReplaceFormat*)

*expression* Required. An expression that returns a **Range** object.

**What** Required **Variant**. The string you want Microsoft Excel to search for.

**Replacement** Required **Variant**. The replacement string.

**LookAt** Optional **Variant**. Can be one of the following **XILookAt** constants: **xlWhole** or **xlPart**.

**SearchOrder** Optional **Variant**. Can be one of the following **XISearchOrder** constants: **xlByRows** or **xlByColumns**.

**MatchCase** Optional **Variant**. **True** to make the search case sensitive.

**MatchByte** Optional **Variant**. You can use this argument only if you've selected or installed double-byte language support in Microsoft Excel. **True** to have double-byte characters match only double-byte characters. **False** to have double-byte characters match their single-byte equivalents.

**SearchFormat** Optional **Variant**. The search format for the method.

**ReplaceFormat** Optional **Variant**. The replace format for the method.

## Remarks

The settings for *LookAt*, *SearchOrder*, *MatchCase*, and *MatchByte* are saved each time you use this method. If you don't specify values for these arguments the next time you call the method, the saved values are used. Setting these arguments changes the settings in the **Find** dialog box, and changing the settings in the **Find** dialog box changes the saved values that are used if you omit the arguments. To avoid problems, set these arguments explicitly each time you use this method.



[Replace method as it applies to the \*\*WorksheetFunction\*\* object.](#)

Replaces part of a text string, based on the number of characters you specify, with a different text string.

*expression*.**Replace**(*Arg1*, *Arg2*, *Arg3*, *Arg4*)

*expression* Required. An expression that returns a [WorksheetFunction](#) object.

**Arg1** Required **String**. Text in which you want to replace some characters.

**Arg2** Required **Double**. The position of the character in **Arg1** that you want to replace with **Arg4**.

**Arg3** Required **Double**. The number of characters in **Arg1** that you want the **Replace** method to replace with **Arg4**.

**Arg4** Required **String**. Text that will replace characters in **Arg1**.

## Example

 [As it applies to the \*\*Range\*\* object.](#)

This example replaces every occurrence of the trigonometric function SIN with the function COS. The replacement range is column A on Sheet1.

```
Worksheets("Sheet1").Columns("A").Replace _  
    What:="SIN", Replacement:="COS", _  
    SearchOrder:=xlByColumns, MatchCase:=True
```

 [As it applies to the \*\*WorksheetFunction\*\* object.](#)

This example replaces abcdef with ac-ef and notifies the user during this process.

```
Sub UseReplace()  
  
    Dim strCurrent As String  
    Dim strReplaced As String  
  
    strCurrent = "abcdef"  
  
    ' Notify user and display current string.  
    MsgBox "The current string is: " & strCurrent  
  
    ' Replace "cd" with "-".  
    strReplaced = Application.WorksheetFunction.Replace _  
        (Arg1:=strCurrent, Arg2:=3, _  
        Arg3:=2, Arg4:="-")  
  
    ' Notify user and display replaced string.  
    MsgBox "The replaced string is: " & strReplaced  
  
End Sub
```



# ReplaceNode Method

Replaces a target diagram node with the source diagram node. The target diagram node is deleted, and the source diagram node, including any of its child nodes, are moved to where the target diagram node was.

*expression*.**ReplaceNode**(*pTargetNode*)

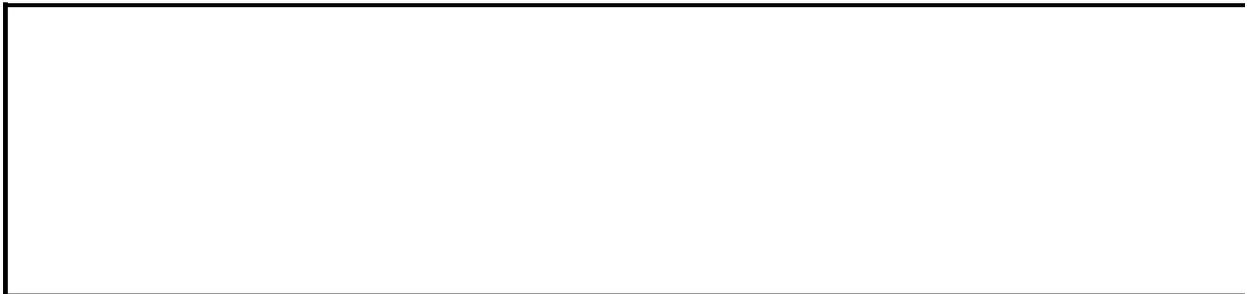
*expression* Required. An expression that returns one of the objects in the Applies To list.

*pTargetNode* Required **DiagramNode** object. The target diagram node to be replaced.

## Example

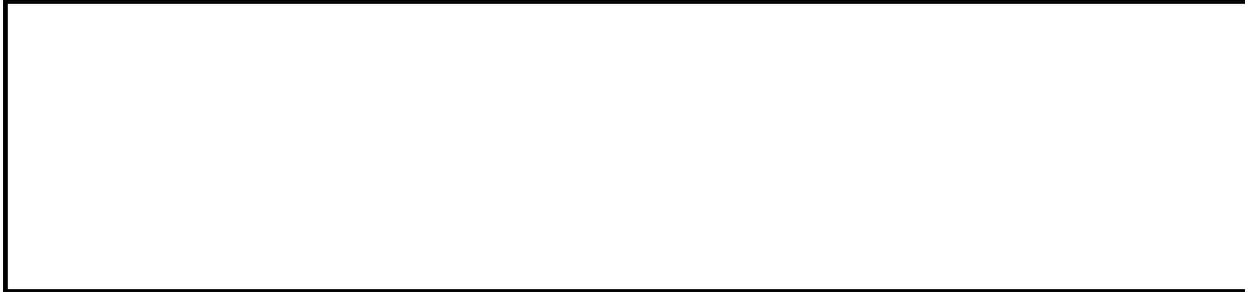
The following example replaces the last diagram node of a newly-created diagram with the second node.

```
Sub ReplaceNode()  
  
    Dim nodRoot As DiagramNode  
    Dim nodPrev As DiagramNode  
    Dim shDiagram As Shape  
    Dim intCount As Integer  
  
    Set shDiagram = ActiveSheet.Shapes.AddDiagram _  
        (Type:=msoDiagramRadial, Left:=10, Top:=15, _  
        Width:=400, Height:=475)  
    Set nodRoot = shDiagram.DiagramNode.Children.AddNode  
  
    ' Add 3 child nodes to the root node.  
    For intCount = 1 To 3  
        nodRoot.Children.AddNode  
    Next  
  
    ' The second node will replace the last node.  
    nodRoot.Children.Item(2).ReplaceNode pTargetNode:=nodRoot.DiagramNode.Children.Item(3)  
  
    ' The count will be 3 since the replaced node was deleted.  
    MsgBox nodRoot.Diagram.Nodes.Count  
  
End Sub
```



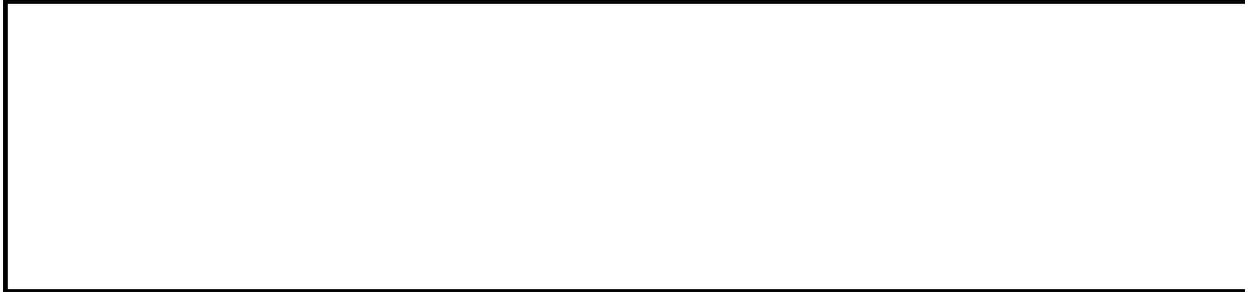
# Reply Method

You have requested Help for a Visual Basic keyword used only on the Macintosh. For information about this keyword, consult the language reference Help included with Microsoft Office Macintosh Edition.



# ReplyAll Method

You have requested Help for a Visual Basic keyword used only on the Macintosh. For information about this keyword, consult the language reference Help included with Microsoft Office Macintosh Edition.



# ReplyWithChanges Method

Sends an e-mail message to the author of a workbook that has been sent out for review, notifying them that a reviewer has completed review of the workbook.

*expression*.**ReplyWithChanges**(*ShowMessage*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**ShowMessage** Optional **Variant**. **False** does not display the message. **True** displays the message.

## Remarks

Use the [SendForReview](#) method to start a collaborative review of a workbook. If the **ReplyWithChanges** method is executed on a workbook that is not part of a collaborative review cycle, the user will receive an error.

## Example

This example automatically sends a notification to the author of a review workbook that a reviewer has completed a review, without first displaying the e-mail message to the reviewer. This example assumes that the active workbook is part of a collaborative review cycle.

```
Sub ReplyMsg()
```

```
    ActiveWorkbook.ReplyWithChanges ShowMessage:=False
```

```
End Sub
```



# RerouteConnections Method

Reroutes connectors so that they take the shortest possible path between the shapes they connect. To do this, the **RerouteConnections** method may detach the ends of a connector and reattach them to different connecting sites on the connected shapes.

This method reroutes all connectors attached to the specified shape; if the specified shape is a connector, it's rerouted.

*expression*.**RerouteConnections**

*expression* Required. An expression that returns a **Shape** or **ShapeRange** object.

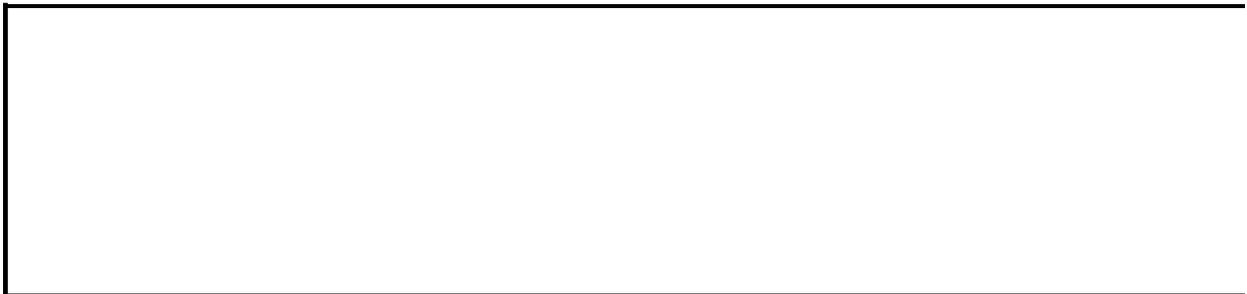
## Remarks

If this method is applied to a connector, only that connector will be rerouted. If this method is applied to a connected shape, all connectors to that shape will be rerouted.

## Example

This example adds two rectangles to myDocument, connects them with a curved connector, and then reroutes the connector so that it takes the shortest possible path between the two rectangles. Note that the **RerouteConnections** method adjusts the size and position of the connector and determines which connecting sites it attaches to, so the values you initially specify for the **ConnectionSite** arguments used with the **BeginConnect** and **EndConnect** methods are irrelevant.

```
Set myDocument = Worksheets(1)
Set s = myDocument.Shapes
Set firstRect = s.AddShape(msoShapeRectangle, _
    100, 50, 200, 100)
Set secondRect = s.AddShape(msoShapeRectangle, _
    300, 300, 200, 100)
Set newConnector = s.AddConnector(msoConnectorCurve, _
    0, 0, 100, 100)
With newConnector.ConnectorFormat
    .BeginConnect firstRect, 1
    .EndConnect secondRect, 1
End With
newConnector.RerouteConnections
```



# Reset Method

Resets the routing slip so that a new routing can be initiated with the same slip (using the same recipient list and delivery information). The routing must be completed before you use this method. Using this method at other times causes an error.

*expression*.**Reset**

*expression* Required. An expression that returns a **RoutingSlip** object.

## Example

This example resets the routing slip for Book1.xls if routing has been completed.

```
With Workbooks("BOOK1.XLS").RoutingSlip
    If .Status = xlRoutingComplete Then
        .Reset
    Else
        MsgBox "Cannot reset routing; not yet complete"
    End If
End With
```



# ResetAllPageBreaks Method

Resets all page breaks on the specified worksheet.

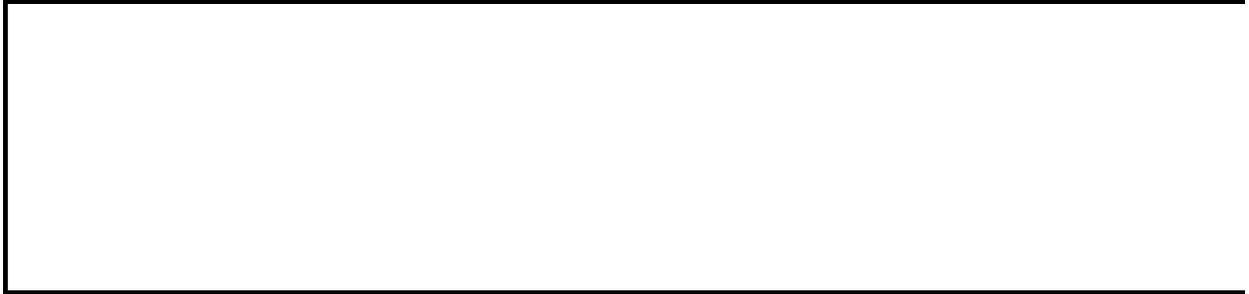
*expression*.**ResetAllPageBreaks()**

*expression* Required. An expression that returns a **Worksheet** object.

## Example

This example resets all page breaks on worksheet one.

```
worksheets(1).ResetAllPageBreaks
```



# ResetColors Method

Resets the color palette to the default colors.

*expression*.**ResetColors**

*expression* Required. An expression that returns a **Workbook** object.

## Example

This example resets the color palette in the active workbook.

`ActiveWorkbook.ResetColors`



# ResetPositionsSideBySide Method

Resets the position of two worksheet windows that are being compared side by side.

*expression*.**ResetPositionsSideBySide()**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Use the **ResetPositionsSideBySide** method to reset the position of two worksheets that are being compared side by side. For example, if a user minimizes or maximizes one of the two worksheet windows being compared, the **ResetPositionsSideBySide** method resets the display so that the two windows are displayed side by side again.

---

# ResetRotation Method

Resets the extrusion rotation around the x-axis and the y-axis to 0 (zero) so that the front of the extrusion faces forward. This method doesn't reset the rotation around the z-axis.

*expression*.**ResetRotation**

*expression* Required. An expression that returns a **ThreeDFormat** object.

## Remarks

To set the extrusion rotation around the x-axis and the y-axis to anything other than 0 (zero), use the [RotationX](#) and [RotationY](#) properties of the **ThreeDFormat** object. To set the extrusion rotation around the z-axis, use the [Rotation](#) property of the [Shape](#) object that represents the extruded shape.

## Example

This example resets the rotation around the x-axis and the y-axis to 0 (zero) for the extrusion of shape one on myDocument.

```
Set myDocument = Worksheets(1)  
myDocument.Shapes(1).ThreeD.ResetRotation
```



# ResetTimer Method

Resets the refresh timer for the specified query table or PivotTable report to the last interval you set using the [RefreshPeriod](#) property.

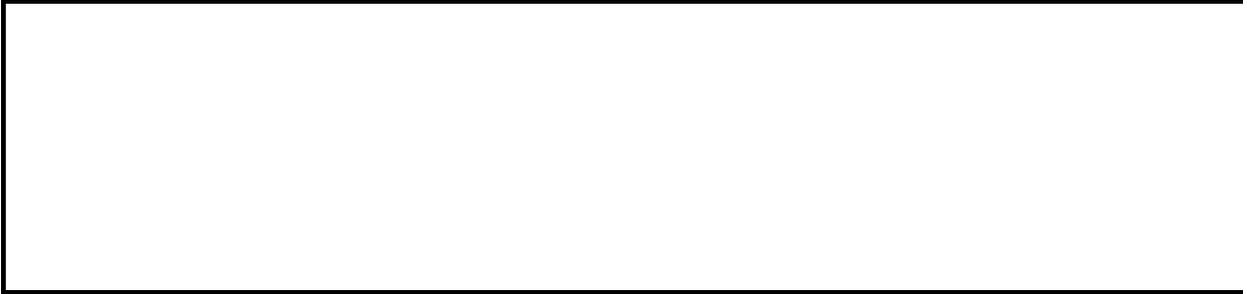
*expression*.**ResetTimer**

*expression* An expression that returns a [PivotCache](#) or [QueryTable](#) object.

## Example

This example resets the refresh timer for the first query table on the active worksheet.

```
ActiveSheet.QueryTables(1).ResetTimer
```



# Resize Method

The **Resize** method allows a **ListObject** object to be resized over a new range. No cells are inserted or moved. Returns **Nothing**.

*expression*.**Resize**(*Range*)

*expression* Required. An expression that returns a **ListObject** object.

*Range* Required **Range**.

## Remarks

For lists that are linked to a server that is running Microsoft Windows SharePoint Services, you can resize the list using this method by providing a **Range** argument that differs from the current range of the ListObject only in the number of rows it contains. Attempting to resize lists linked to Windows SharePoint Services by adding or deleting columns (in the **Range** argument) results in a run-time error.

The following example uses the **Resize** method to resize the default **ListObject** object in Sheet1 of the active workbook.

```
Sub ResizeList()  
    Dim wrksht As Worksheet  
    Dim objListObj As ListObject  
  
    Set wrksht = ActiveWorkbook.Worksheets("Sheet1")  
    Set objListObj = wrksht.ListObjects(1)  
  
    objListObj.Resize Range("A1:B10")  
End Sub
```



# RestartServers Method

Reconnects to a real-time data server (RTD).

*expression*.**RestartServers()**

*expression* Required. An expression that returns an [RTD](#) object.



# Route Method

Routes the workbook, using the workbook's current routing slip.

*expression*.**Route**

*expression* Required. An expression that returns a **Workbook** object.

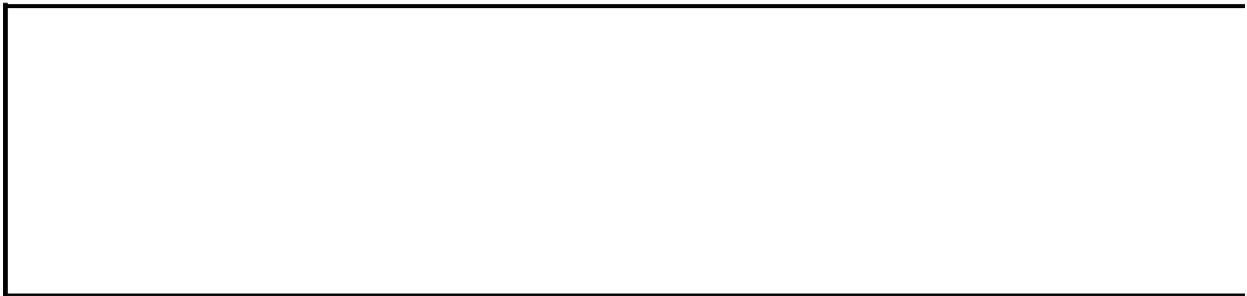
## Remarks

Routing a workbook sets the **Routed** property to **True**.

## Example

This example creates a routing slip for Book1.xls and then sends the workbook to three recipients, one after another.

```
Workbooks("BOOK1.XLS").HasRoutingSlip = True
With Workbooks("BOOK1.XLS").RoutingSlip
    .Delivery = xlOneAfterAnother
    .Recipients = Array("Adam Bendel", _
        "Jean Selva", "Bernard Gabor")
    .Subject = "Here is BOOK1.XLS"
    .Message = "Here is the workbook. What do you think?"
End With
Workbooks("BOOK1.XLS").Route
```



# RowDifferences Method

Returns a [Range](#) object that represents all the cells whose contents are different from those of the comparison cell in each row.

*expression*.**RowDifferences**(*Comparison*)

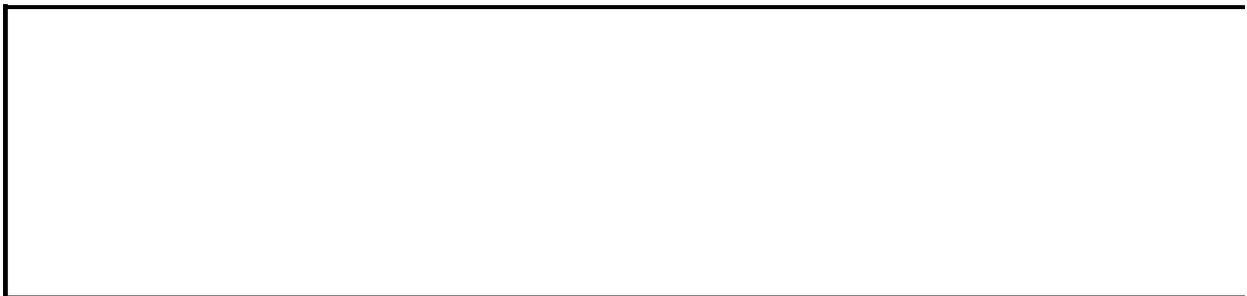
*expression* Required. An expression that returns a range containing the cells to be compared.

**Comparison** Required **Variant**. A single cell to compare with the specified range.

## Example

This example selects the cells in row one on Sheet1 whose contents are different from those of cell D1.

```
Worksheets("Sheet1").Activate  
Set c1 = ActiveSheet.Rows(1).RowDifferences( _  
    comparison:=ActiveSheet.Range("D1"))  
c1.Select
```



[Show All](#)

# RTD Method

This method connects to a source to receive real-time data.

*expression*.RTD (**progID**, **server**, **topic1**, **topic2**, **topic3**, **topic4**, **topic5**, **topic6**, **topic7**, **topic8**, **topic9**, **topic10**, **topic11**, **topic12**, **topic13**, **topic14**, **topic15**, **topic16**, **topic17**, **topic18**, **topic19**, **topic20**, **topic21**, **topic22**, **topic23**, **topic24**, **topic25**, **topic26**, **topic27**, **topic28**)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**progID** Required **Variant**. A string representing the real-time server programmatic identifier.

**server** Required **Variant**. A server name, **Null** string or **vbNullString** constant.

**topic1** Required **Variant**. A **String** representing a topic.

**topic2-topic28** Optional **Variant**. A **String** representing a topic.

## Remarks

The *server* argument is required in [Visual Basic for Applications \(VBA\)](#), even though it can be omitted within a worksheet.

--

[Show All](#)

# Run Method



Runs the Microsoft Excel macro at this location. The range must be on a macro sheet.

*expression*.**Run**(*Arg1*, *Arg2*, *Arg3*, *Arg4*, *Arg5*, *Arg6*, *Arg7*, *Arg8*, *Arg9*, *Arg10*, *Arg11*, *Arg12*, *Arg13*, *Arg14*, *Arg15*, *Arg16*, *Arg17*, *Arg18*, *Arg19*, *Arg20*, *Arg21*, *Arg22*, *Arg23*, *Arg24*, *Arg25*, *Arg26*, *Arg27*, *Arg28*, *Arg29*, *Arg30*)

*expression* Required. An expression that returns a [Range](#) object.

**Arg1-Arg30** Optional **Variant**. The arguments that should be passed to the function.



Runs a macro or calls a function. This can be used to run a macro written in Visual Basic or the Microsoft Excel macro language, or to run a function in a DLL or XLL.

*expression*.**Run**(*Macro*, *Arg1*, *Arg2*, *Arg3*, *Arg4*, *Arg5*, *Arg6*, *Arg7*, *Arg8*, *Arg9*, *Arg10*, *Arg11*, *Arg12*, *Arg13*, *Arg14*, *Arg15*, *Arg16*, *Arg17*, *Arg18*, *Arg19*, *Arg20*, *Arg21*, *Arg22*, *Arg23*, *Arg24*, *Arg25*, *Arg26*, *Arg27*, *Arg28*, *Arg29*, *Arg30*)

*expression* Required. An expression that returns an [Application](#) object.

**Macro** Optional **Variant**. The macro to run. This can be either a string with the macro name, a **Range** object indicating where the function is, or a register ID for a registered DLL (XLL) function. If a string is used, the string will be evaluated in the context of the active sheet.

**Arg1-Arg30** Optional **Variant**. The arguments that should be passed to the function.

## Remarks

You cannot use named arguments with this method. Arguments must be passed by position.

The **Run** method returns whatever the called macro returns. Objects passed as arguments to the macro are converted to values (by applying the **Value** property to the object). This means that you cannot pass objects to macros by using the **Run** method.

## Example

This example shows how to call the function macro My\_Func\_Sum, which is defined on the macro sheet Mycustom.xlm (the macro sheet must be open). The function takes two numeric arguments (1 and 5, in this example).

```
mySum = Application.Run("MYCUSTOM.XLM!My_Func_Sum", 1, 5)  
MsgBox "Macro result: " & mySum
```



[Show All](#)

# RunAutoMacros Method

Runs the Auto\_Open, Auto\_Close, Auto\_Activate, or Auto\_Deactivate macro attached to the workbook. This method is included for backward compatibility. For new Visual Basic code, you should use the Open, Close, Activate and Deactivate events instead of these macros.

*expression*.**RunAutoMacros**(*Which*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*Which* Required [XlRunAutoMacro](#).

XlRunAutoMacro can be one of these XlRunAutoMacro constants.

**xlAutoActivate**. Auto\_Activate macros

**xlAutoClose**. Auto\_Close macros

**xlAutoDeactivate**. Auto\_Deactivate macros

**xlAutoOpen**. Auto\_Open macros

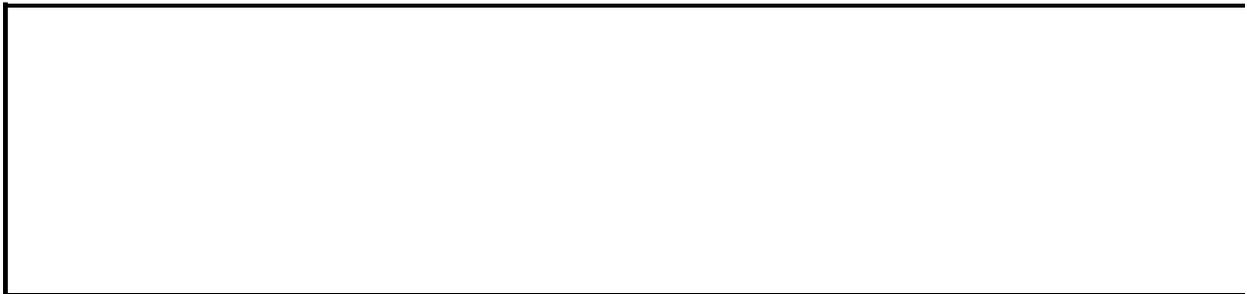
## Example

This example opens the workbook Analysis.xls and then runs its Auto\_Open macro.

```
Workbooks.Open "ANALYSIS.XLS"  
ActiveWorkbook.RunAutoMacros xlAutoOpen
```

This example runs the Auto\_Close macro for the active workbook and then closes the workbook.

```
With ActiveWorkbook  
    .RunAutoMacros xlAutoClose  
    .Close  
End With
```



# Save Method

Saves changes to the specified workbook.

*expression*.**Save**

*expression* Required. An expression that returns a **Workbook** object.

## Remarks

To open a workbook file, use the [Open](#) method.

To mark a workbook as saved without writing it to a disk, set its [Saved](#) property to **True**.

The first time you save a workbook, use the [SaveAs](#) method to specify a name for the file.

## Example

This example saves the active workbook.

```
ActiveWorkbook.Save
```

This example saves all open workbooks and then closes Microsoft Excel.

```
For Each w In Application.Workbooks  
    w.Save  
Next w  
Application.Quit
```



[Show All](#)

# SaveAs Method

 [SaveAs method as it applies to the \*\*Chart\*\* and \*\*Worksheet\*\* objects.](#)

Saves changes to the chart or worksheet in a different file.

*expression*.**SaveAs**(*FileName*, *FileFormat*, *Password*, *WriteResPassword*, *ReadOnlyRecommended*, *CreateBackup*, *AddToMru*, *TextCodepage*, *TextVisualLayout*, *Local*)

*expression* Required. An expression that returns one of the above objects.

**Filename** Optional **Variant**. A string that indicates the name of the file to be saved. You can include a full path; if you don't, Microsoft Excel saves the file in the current folder.

**FileFormat** Optional **Variant**. The file format to use when you save the file. For a list of valid choices, see the [FileFormat](#) property. For an existing file, the default format is the last file format specified; for a new file, the default is the format of the version of Excel being used.

**Password** Optional **Variant**. A case-sensitive string (no more than 15 characters) that indicates the protection password to be given to the file.

**WriteResPassword** Optional **Variant**. A string that indicates the write-reservation password for this file. If a file is saved with the password and the password isn't supplied when the file is opened, the file is opened as read-only.

**ReadOnlyRecommended** Optional **Variant**. **True** to display a message when the file is opened, recommending that the file be opened as read-only.

**CreateBackup** Optional **Variant**. **True** to create a backup file.

**AddToMru** Optional **Variant**. **True** to add this workbook to the list of recently used files. The default value is **False**.

**TextCodePage** Optional **Variant**. Not used in U.S. English Microsoft Excel.

**TextVisualLayout** Optional **Variant**. Not used in U.S. English Microsoft Excel.

**Local** Optional **Variant**. **True** saves files against the language of Microsoft Excel (including control panel settings). **False** (default) saves files against the language of [Visual Basic for Applications \(VBA\)](#) (which is typically US English unless the VBA project where Workbooks.Open is run from is an old internationalized XL5/95 VBA project).

 [SaveAs method as it applies to the Workbook object.](#)

Saves changes to the workbook in a different file.

*expression.SaveAs(FileName, FileFormat, Password, WriteResPassword, ReadOnlyRecommended, CreateBackup, AccessMode, ConflictResolution, AddToMru, TextCodepage, TextVisualLayout, Local)*

*expression* Required. An expression that returns one of the above objects.

**Filename** Optional **Variant**. A string that indicates the name of the file to be saved. You can include a full path; if you don't, Microsoft Excel saves the file in the current folder.

**FileFormat** Optional **Variant**. The file format to use when you save the file. For a list of valid choices, see the [FileFormat](#) property. For an existing file, the default format is the last file format specified; for a new file, the default is the format of the version of Excel being used.

**Password** Optional **Variant**. A case-sensitive string (no more than 15 characters) that indicates the protection password to be given to the file.

**WriteResPassword** Optional **Variant**. A string that indicates the write-reservation password for this file. If a file is saved with the password and the password isn't supplied when the file is opened, the file is opened as read-only.

**ReadOnlyRecommended** Optional **Variant**. **True** to display a message when the file is opened, recommending that the file be opened as read-only.

**CreateBackup** Optional **Variant**. **True** to create a backup file.

**AccessMode** Optional [XISaveAsAccessMode](#).

XISaveAsAccessMode can be one of these XISaveAsAccessMode constants.

**xlExclusive** (exclusive mode)

**xlNoChange** *default* (don't change the access mode)

**xlShared** (share list)

If this argument is omitted, the access mode isn't changed. This argument is ignored if you save a shared list without changing the file name. To change the access mode, use the [ExclusiveAccess](#) method.

**ConflictResolution** Optional [XISaveConflictResolution](#).

XISaveConflictResolution can be one of these XISaveConflictResolution constants.

**xlUserResolution** (display the conflict-resolution dialog box)

**xlLocalSessionChanges** (automatically accept the local user's changes)

**xlOtherSessionChanges** (accept other changes instead of the local user's changes)

If this argument is omitted, the conflict-resolution dialog box is displayed.

**AddToMru** Optional **Variant**. **True** to add this workbook to the list of recently used files. The default value is **False**.

**TextCodePage** Optional **Variant**. Not used in U.S. English Microsoft Excel.

**TextVisualLayout** Optional **Variant**. Not used in U.S. English Microsoft Excel.

**Local** Optional **Variant**. **True** saves files against the language of Microsoft Excel (including control panel settings). **False** (default) saves files against the language of [Visual Basic for Applications \(VBA\)](#) (which is typically US English unless the VBA project where Workbooks.Open is run from is an old internationalized XL5/95 VBA project).

## Example

This example creates a new workbook, prompts the user for a file name, and then saves the workbook.

```
Set NewBook = Workbooks.Add  
Do  
    fName = Application.GetSaveAsFilename  
Loop Until fName <> False  
NewBook.SaveAs Filename:=fName
```



# SaveAsODC Method

Saves the PivotTable cache source as an Microsoft Office Data Connection file.

*expression*.**SaveAsODC**(*ODCFileName*, *Description*, *Keywords*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**ODCFileName** Required **String**. Location to save the file.

**Description** Optional **Variant**. Description that will be saved in the file.

**Keywords** Optional **Variant**. Space-separated keywords that can be used to search for this file.

## Example

The following example saves the cache source as an ODC file titled "ODCFile". This example assumes a PivotTable cache exists on the active worksheet.

```
Sub UseSaveAsODC()  
    Application.ActiveWorkbook.PivotCaches.Item(1).SaveAsODC ("ODCFi  
End Sub
```



# SaveAsXMLData Method

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Exports the data that has been mapped to the specified XML schema map to an XML data file.

*expression*.**SaveAsXMLData**(*Filename*, *Map*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Filename** Required **String**. A string that indicates the name of the file to be saved. You can include a full path; if you don't, Microsoft Excel saves the file in the current folder.

**Map** Required [XmlMap](#) object. The schema map to apply to the data.

## Remarks

This method will result in a run-time error if Excel cannot export data with the specified schema map. To check whether Excel can use the specified schema map to export data, use the [IsExportable](#) property.

## Example

The following example verifies that Excel can use the schema map "Customer" to export data, and then exports the data mapped to the "Customer" schema map to a file named "Customer Data.xml".

```
Sub ExportAsXMLData()  
    Dim objMapToExport As XmlMap  
  
    Set objMapToExport = ActiveWorkbook.XmlMaps("Customer")  
  
    If objMapToExport.IsExportable Then  
  
        ActiveWorkbook.SaveAsXMLData "Customer Data.xml", objMapToEx  
    Else  
        MsgBox "Cannot use " & objMapToExport.Name & _  
            "to export the contents of the worksheet to XML data."  
    End If  
End Sub
```



# SaveCopyAs Method

Saves a copy of the workbook to a file but doesn't modify the open workbook in memory.

*expression*.**SaveCopyAs**(*Filename*)

*expression* Required. An expression that returns a **Workbook** object.

**Filename** Required. Specifies the file name for the copy.

## Example

This example saves a copy of the active workbook.

```
ActiveWorkbook.SaveCopyAs "C:\TEMP\XXXX.XLS"
```



# SaveWorkspace Method

Saves the current workspace.

*expression*.**SaveWorkspace**(*Filename*)

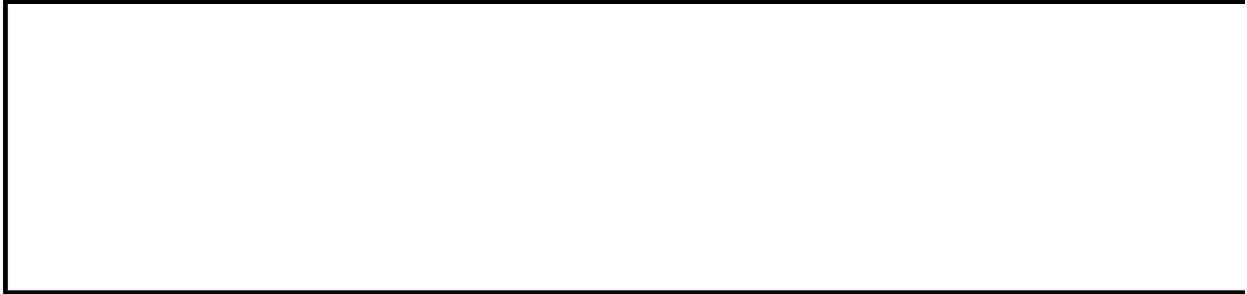
*expression* Required. An expression that returns an **Application** object.

**Filename** Optional **Variant**. The saved file name.

## Example

This example saves the current workspace as "saved workspace.xlw".

```
Application.SaveWorkspace "saved workspace"
```



[Show All](#)

# ScaleHeight Method

Scales the height of the shape by a specified factor. For pictures and OLE objects, you can indicate whether you want to scale the shape relative to the original or the current size. Shapes other than pictures and OLE objects are always scaled relative to their current height.

*expression*.**ScaleHeight**(*Factor*, *RelativeToOriginalSize*, *Scale*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Factor** Required **Single**. Specifies the ratio between the height of the shape after you resize it and the current or original height. For example, to make a rectangle 50 percent larger, specify 1.5 for this argument.

**RelativeToOriginalSize** Required **MsoTriState**. **msoTrue** to scale the shape relative to its original size. **msoFalse** to scale it relative to its current size. You can specify **msoTrue** for this argument only if the specified shape is a picture or an OLE object.

MsoTriState can be one of these MsoTriState constants.

**msoCTrue** Does not apply to this property.

**msoFalse** Scale the shape relative to its current size.

**msoTriStateMixed** Does not apply to this property.

**msoTriStateToggle** Does not apply to this property.

**msoTrue** Scale the shape relative to its original size.

**Scale** Optional **MsoScaleFrom**. Specifies which part of the shape retains its position when the shape is scaled.

MsoScaleFrom can be one of these MsoScaleFrom constants.

**msoScaleFromBottomRight**

**msoScaleFromMiddle**

**msoScaleFromTopLeft** *default*

## Example

This example scales all pictures and OLE objects on myDocument to 175 percent of their original height and width, and it scales all other shapes to 175 percent of their current height and width.

```
Set myDocument = Worksheets(1)
For Each s In myDocument.Shapes
  Select Case s.Type
    Case msoEmbeddedOLEObject, _
         msoLinkedOLEObject, _
         msoOLEControlObject, _
         msoLinkedPicture, msoPicture
      s.ScaleHeight 1.75, msoTrue
      s.ScaleWidth 1.75, msoTrue
    Case Else
      s.ScaleHeight 1.75, msoFalse
      s.ScaleWidth 1.75, msoFalse
  End Select
Next
```



[Show All](#)

# ScaleWidth Method

Scales the width of the shape by a specified factor. For pictures and OLE objects, you can indicate whether you want to scale the shape relative to the original or the current size. Shapes other than pictures and OLE objects are always scaled relative to their current width.

*expression*.**ScaleWidth**(*Factor*, *RelativeToOriginalSize*, *Scale*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Factor** Required **Single**. Specifies the ratio between the width of the shape after you resize it and the current or original width. For example, to make a rectangle 50 percent larger, specify 1.5 for this argument.

**RelativeToOriginalSize** Required **MsoTriState**. **False** to scale it relative to its current size. You can specify **True** for this argument only if the specified shape is a picture or an OLE object.

MsoTriState can be one of these MsoTriState constants.

**msoCTrue** Does not apply to this property.

**msoFalse** To scale it relative to its current size.

**msoTriStateMixed** Does not apply to this property.

**msoTriStateToggle** Does not apply to this property.

**msoTrue** Can only use this argument if the specified shape is a picture or an OLE object.

**Scale** Optional **MsoScaleFrom**. Specifies which part of the shape retains its position when the shape is scaled.

MsoScaleFrom can be one of these MsoScaleFrom constants.

**msoScaleFromBottomRight**

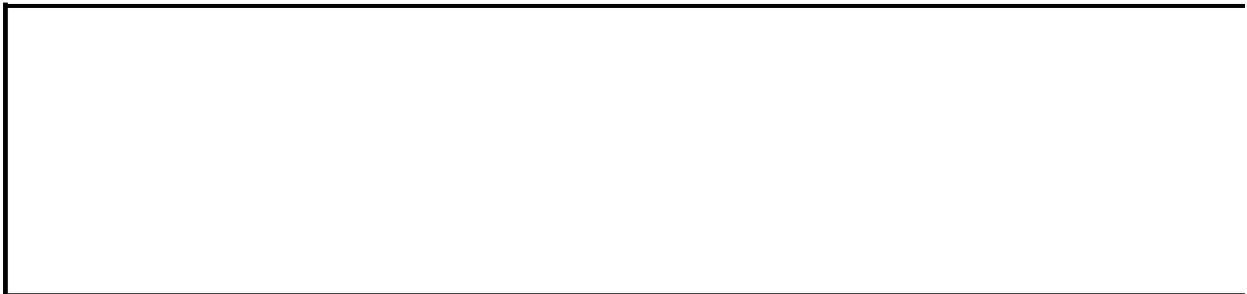
**msoScaleFromMiddle**

**msoScaleFromTopLeft** *default*

## Example

This example scales all pictures and OLE objects on myDocument to 175 percent of their original height and width, and it scales all other shapes to 175 percent of their current height and width.

```
Set myDocument = Worksheets(1)
For Each s In myDocument.Shapes
  Select Case s.Type
    Case msoEmbeddedOLEObject, _
      msoLinkedOLEObject, _
      msoOLEControlObject, _
      msoLinkedPicture, msoPicture
      s.ScaleHeight 1.75, msoTrue
      s.ScaleWidth 1.75, ,msoTrue
    Case Else
      s.ScaleHeight 1.75, msoFalse
      s.ScaleWidth 1.75, msoFalse
  End Select
Next
```



# Scenarios Method

Returns an object that represents either a single scenario (a [Scenario](#) object) or a collection of scenarios (a [Scenarios](#) object) on the worksheet.

*expression.Scenarios(Index)*

*expression* Required. An expression that returns a **Worksheet** object.

**Index** Optional **Variant**. The name or number of the scenario. Use an array to specify more than one scenario.

## Example

This example sets the comment for the first scenario on Sheet1.

```
Worksheets("Sheet1").Scenarios(1).Comment = _  
    "Worst-case July 1993 sales"
```



# ScrollIntoView Method

Scrolls the document window so that the contents of a specified rectangular area are displayed in either the upper-left or lower-right corner of the document window or pane (depending on the value of the **Start** argument).

*expression.ScrollIntoView(Left, Top, Width, Height, Start)*

*expression* An expression that returns a [Pane](#) or [Window](#) object.

**Left** Required **Long**. The horizontal position of the rectangle (in points) from the left edge of the document window or pane.

**Top** Required **Long**. The vertical position of the rectangle (in points) from the top of the document window or pane.

**Width** Required **Long**. The width of the rectangle, in points.

**Height** Required **Long**. The height of the rectangle, in points.

**Start** Optional **Variant**. **True** to have the upper-left corner of the rectangle appear in the upper-left corner of the document window or pane. **False** to have the lower-right corner of the rectangle appear in the lower-right corner of the document window or pane. The default value is **True**.

## Remarks

The *Start* argument is useful for orienting the screen display when the rectangle is larger than the document window or pane.

## Example

This example defines a 100-by-200-pixel rectangle in the active document window, positioned 20 pixels from the top of the window and 50 pixels from the left edge of the window. The example then scrolls the document up and to the left so that the upper-left corner of the rectangle is aligned with the upper-left corner of the window.

```
ActiveWindow.ScrollIntoView _  
    Left:=50, Top:=20, _  
    Width:=100, Height:=200
```



# ScrollWorkbookTabs Method

Scrolls through the workbook tabs at the bottom of the window. Doesn't affect the active sheet in the workbook.

*expression*.ScrollWorkbookTabs(*Sheets*, *Position*)

*expression* Required. An expression that returns a **Window** object.

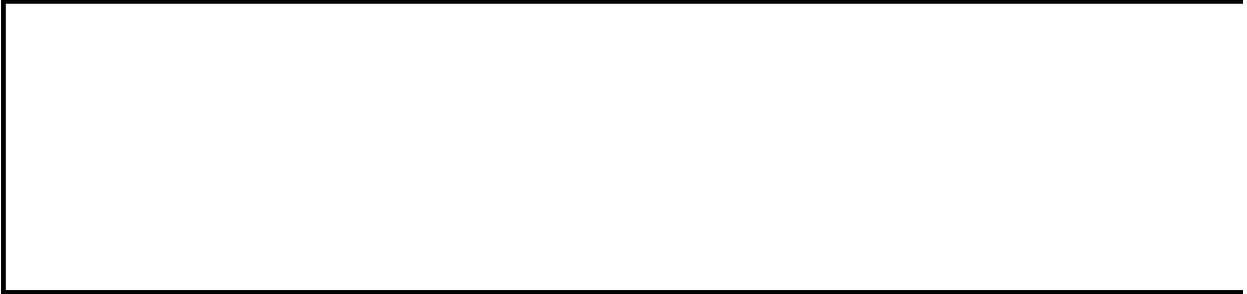
**Sheets** Optional **Variant**. The number of sheets to scroll by. Use a positive number to scroll forward, a negative number to scroll backward, or 0 (zero) to not scroll at all. You must specify **Sheets** if you don't specify **Position**.

**Position** Optional **Variant**. Use **xlFirst** to scroll to the first sheet, or use **xlLast** to scroll to the last sheet. You must specify **Position** if you don't specify **Sheets**.

## Example

This example scrolls through the workbook tabs to the last sheet in the workbook.

```
ActiveWindow.ScrollWorkbookTabs position:=xlLast
```



[Show All](#)

# Select Method

[Select method as it applies to the \*\*ChartObject\*\*, \*\*ChartObjects\*\*, \*\*OLEObject\*\*, and \*\*OLEObjects\*\* objects.](#)

Selects the object.

*expression*.**Select(Replace)**

*expression* Required. An expression that returns one of the above objects.

**Replace** Optional **Variant**. **True** to replace the current selection with the specified object. **False** to extend the current selection to include any previously selected objects and the specified object.

[Select method as it applies to the \*\*DataTable\*\* and \*\*LeaderLines\*\* objects.](#)

Selects the object.

*expression*.**Select**

*expression* Required. An expression that returns one of the above objects.

[Select method as it applies to the \*\*Chart\*\*, \*\*Charts\*\*, \*\*Shape\*\*, \*\*ShapeRange\*\*, \*\*Sheets\*\*, \*\*Worksheet\*\*, and \*\*Worksheets\*\* objects.](#)

Selects the object.

*expression*.**Select(Replace)**

*expression* Required. An expression that returns one of the above objects.

**Replace** Optional **Variant**. The object to replace.

[Select method as it applies to all other objects in the \*\*Applies To\*\* list.](#)

Selects the object.

*expression*.**Select**

*expression* Required. An expression that returns all other objects in the Applies To list.

## Remarks

To select a cell or a range of cells, use the **Select** method. To make a single cell the active cell, use the [Activate](#) method.

## Example

This example selects cells A1:B3 on Sheet1.

```
Worksheets("Sheet1").Activate  
Range("A1:B3").Select
```



# SelectAll Method

Selects all the shapes in the specified **CanvasShapes**, **DiagramNodeChildren**, **DiagramNodes**, or **Shapes** collection.

*expression*.**SelectAll**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example selects all the shapes on myDocument and creates a **ShapeRange** collection containing all the shapes.

```
Set myDocument = Worksheets(1)  
myDocument.Shapes.SelectAll
```

```
Set sr = Selection.ShapeRange
```



# SendFaxOverInternet Method

Sends a worksheet as a fax to the specified recipients.

*expression*.**SendFaxOverInternet**(*Recipients*, *Subject*, *ShowMessage*)

*expression* Required. An expression that returns a [Workbook](#) object.

**Recipients** Optional **Variant**. A **String** that represents the fax numbers and e-mail addresses of the people to whom the fax will be sent. Separate multiple recipients with a semicolon.

**Subject** Optional **Variant**. A **String** that represents the subject line for the faxed document.

**ShowMessage** Optional **Variant**. **True** displays the fax message before sending it. **False** sends the fax without displaying the fax message.

## Remarks

Using the **SendFaxOverInternet** method requires that the fax service is enabled on a user's computer.

The format used for specifying fax numbers in the **Recipients** parameter is either *recipientsfaxnumber@usersfaxprovider* or *recipientsname@recipientsfaxnumber*. You can access the user's fax provider information using the following registry path:

```
HKEY_CURRENT_USER\Software\Microsoft\Office\11.0\Common\Services\Fax
```

Use the value of the FaxAddress key at this registry path to determine the format to use for a recipient.

## Example

The following example sends a fax to the fax service provider, which will then fax the message to the recipient.

```
ActiveWorkbook.SendFaxOverInternet _  
    "14255550101@consolidatedmessenger.com", _  
    "For your review", True
```



# SendForReview Method

Sends a workbook in an e-mail message for review to the specified recipients.

*expression*.**SendForReview**(*Recipients*, *Subject*, *ShowMessage*, *IncludeAttachment*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Recipients** Optional **Variant**. A string that lists the people to whom to send the message. These can be unresolved names and aliases in an e-mail phone book or full e-mail addresses. Separate multiple recipients with a semicolon (;). If left blank and **ShowMessage** is **False**, you will receive an error message, and the message will not be sent.

**Subject** Optional **Variant**. A string for the subject of the message. If left blank, the subject will be: Please review "*filename*".

**ShowMessage** Optional **Variant**. A **Boolean** value that indicates whether the message should be displayed when the method is executed. The default value is **True**. If set to **False**, the message is automatically sent to the recipients without first showing the message to the sender.

**IncludeAttachment** Optional **Variant**. A **Boolean** value that indicates whether the message should include an attachment or a link to a server location. The default value is **True**. If set to **False**, the document must be stored at a shared location.

## Remarks

The **SendForReview** method starts a collaborative review cycle. Use the [EndReview](#) method to end a review cycle.

## Example

This example automatically sends the active workbook as an attachment in an e-mail message to the specified recipients.

```
Sub WebReview()  
    ActiveWorkbook.SendForReview _  
        Recipients:="someone@microsoft.com; amy jones; lewjudy", _  
        Subject:="Please review this document.", _  
        ShowMessage:=False, _  
        IncludeAttachment:=True  
End Sub
```



# SendKeys Method

Sends keystrokes to the active application.

*expression*.**SendKeys**(*Keys*, *Wait*)

*expression* Optional. An expression that returns an **Application** object.

**Keys** Required **Variant**. The key or key combination you want to send to the application, as text.

**Wait** Optional **Variant**. **True** to have Microsoft Excel wait for the keys to be processed before returning control to the macro. **False** (or omitted) to continue running the macro without waiting for the keys to be processed.

## Remarks

This method places keystrokes in a key buffer. In some cases, you must call this method before you call the method that will use the keystrokes. For example, to send a password to a dialog box, you must call the **SendKeys** method before you display the dialog box.

The **Keys** argument can specify any single key or any key combined with ALT, CTRL, or SHIFT (or any combination of those keys). Each key is represented by one or more characters, such as "a" for the character a, or "{ENTER}" for the ENTER key.

To specify characters that aren't displayed when you press the corresponding key (for example, ENTER or TAB), use the codes listed in the following table. Each code in the table represents one key on the keyboard.

Key	Code
BACKSPACE	{BACKSPACE} or {BS}
BREAK	{BREAK}
CAPS LOCK	{CAPSLOCK}
CLEAR	{CLEAR}
DELETE or DEL	{DELETE} or {DEL}
DOWN ARROW	{DOWN}
END	{END}
ENTER	~ (tilde)
ENTER (numeric keypad)	{ENTER}
ESC	{ESCAPE} or {ESC}
F1 through F15	{F1} through {F15}
HELP	{HELP}
HOME	{HOME}
INS	{INSERT}
LEFT ARROW	{LEFT}
NUM LOCK	{NUMLOCK}
PAGE DOWN	{PGDN}

PAGE UP	{PGUP}
RETURN	{RETURN}
RIGHT ARROW	{RIGHT}
SCROLL LOCK	{SCROLLLOCK}
TAB	{TAB}
UP ARROW	{UP}

You can also specify keys combined with SHIFT and/or CTRL and/or ALT. To specify a key combined with another key or keys, use the following table.

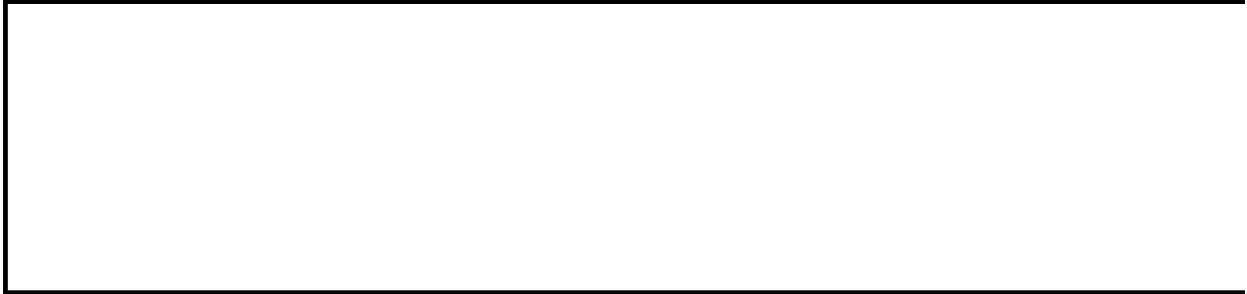
**To combine a key with Precede the key code with**

SHIFT	+ (plus sign)
CTRL	^ (caret)
ALT	% (percent sign)

## Example

This example uses the **SendKeys** method to quit Microsoft Excel.

```
Application.SendKeys("%fx")
```



# SendMail Method

Sends the workbook by using the installed mail system.

*expression*.**SendMail**(*Recipients*, *Subject*, *ReturnReceipt*)

*expression* Required. An expression that returns a **Workbook** object.

**Recipients** Required **Variant**. Specifies the name of the recipient as text, or as an array of text strings if there are multiple recipients. At least one recipient must be specified, and all recipients are added as To recipients.

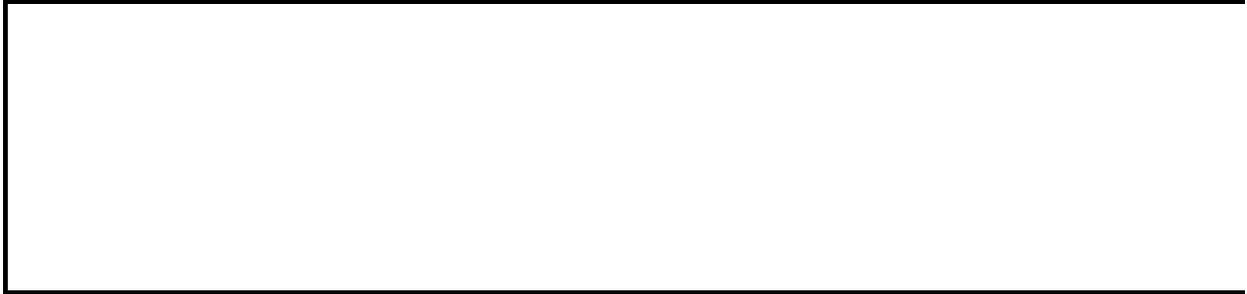
**Subject** Optional **Variant**. Specifies the subject of the message. If this argument is omitted, the document name is used.

**ReturnReceipt** Optional **Variant**. **True** to request a return receipt. **False** to not request a return receipt. The default value is **False**.

## Example

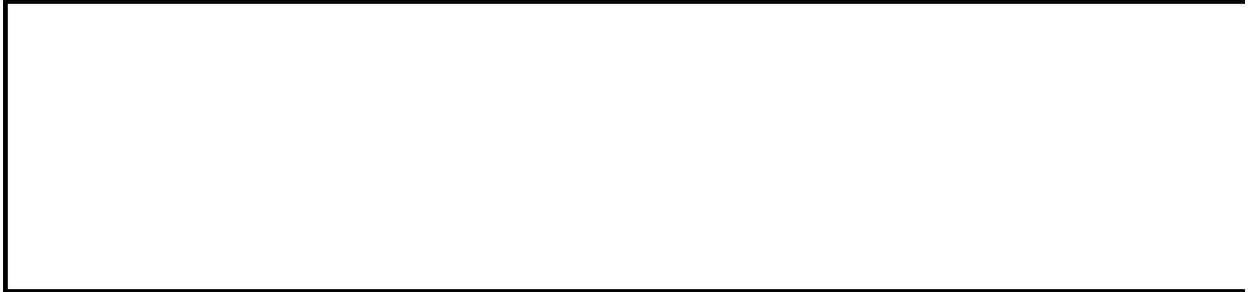
This example sends the active workbook to a single recipient.

```
ActiveWorkbook.SendMail recipients:="Jean Selva"
```



# SendMailer Method

You have requested Help for a Visual Basic keyword used only on the Macintosh. For information about this keyword, consult the language reference Help included with Microsoft Office Macintosh Edition.



# SendToBack Method

Sends the object to the back of the z-order.

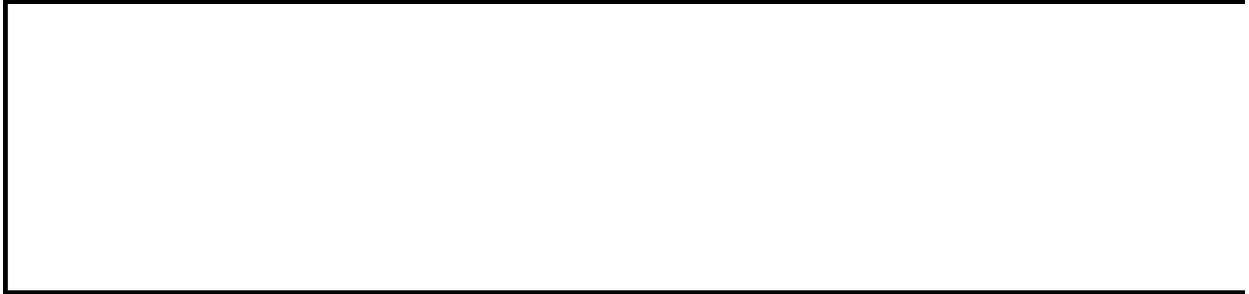
*expression*.**SendToBack**

*expression* Required. An expression that returns an object in the Applies To list.

## Example

This example sends embedded chart one on Sheet1 to the back of the z-order.

```
worksheets("Sheet1").ChartObjects(1).SendToBack
```



# SeriesCollection Method

Returns an object that represents either a single series (a [Series](#) object) or a collection of all the series (a [SeriesCollection](#) collection) in the chart or chart group.

*expression*.SeriesCollection(*Index*)

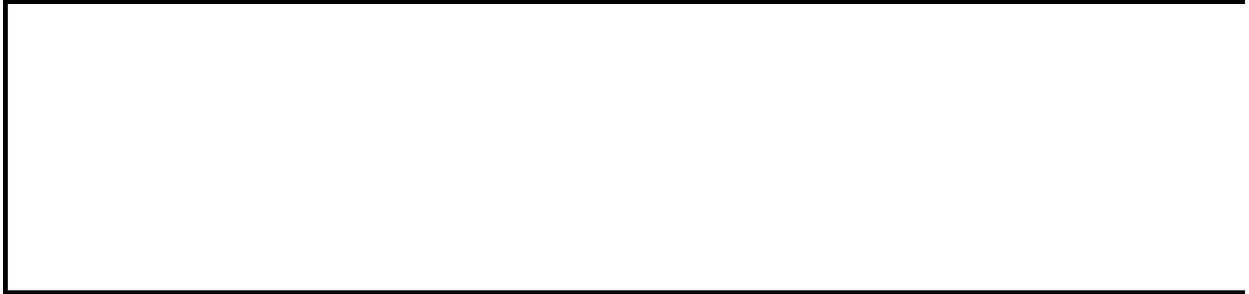
*expression* Required. An expression that returns a **Chart** or **ChartGroup** object.

*Index* Optional **Variant**. The name or number of the series.

## Example

This example turns on data labels for series one in Chart1.

```
Charts("Chart1").SeriesCollection(1).HasDataLabels = True
```



# ServerStart Method

The **ServerStart** method is called immediately after a real-time data server is instantiated. Returns a **Long**; negative value or zero indicates failure to start the server; positive value indicates success.

*expression*.**ServerStart**(*CallbackObject*)

*expression* Required. An expression that returns an **IRtdServer** object.

*CallbackObject* Required **IRTDUpdateEvent** object. The callback object.



# ServerTerminate Method

Terminates the connection to the real-time data server.

*expression*.**ServerTerminate**

*expression* Required. An expression that returns an [IRtdServer](#) object.



# SetBackgroundPicture Method

Sets the background graphic for a worksheet or chart.

*expression*.**SetBackgroundPicture**(*FileName*)

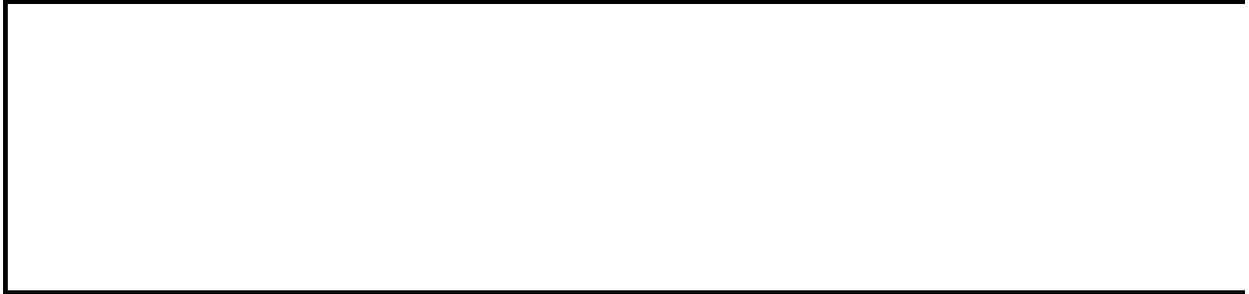
*expression* Required. An expression that returns a **Worksheet** or **Chart** object.

**FileName** Required **String**. The name of the graphic file.

## Example

This example sets the background graphic for worksheet one.

```
Worksheets(1).SetBackgroundPicture "c:\graphics\watermark.gif"
```



# SetDefaultChart Method

Specifies the name of the chart template that Microsoft Excel will use when creating new charts.

*expression*.**SetDefaultChart**(*FormatName*)

*expression* Required. An expression that returns an **Application** object.

**FormatName** Optional **Variant**. Specifies the name of a custom autofilter. This name can be a string naming a custom autofilter, or it can be the special constant **xlBuiltIn** to specify the built-in chart template.

## Example

This example sets the default chart template to the custom autoformat named "Monthly Sales."

```
Application.SetDefaultChart FormatName:="Monthly Sales"
```



[Show All](#)

# SetEditingType Method

Sets the editing type of the node specified by ***Index***. If the node is a control point for a curved segment, this method sets the editing type of the node adjacent to it that joins two segments. Note that, depending on the editing type, this method may affect the position of adjacent nodes.

*expression*.**SetEditingType**(***Index***, ***EditingType***)

*expression* Required. An expression that returns one of the objects in the Applies To list.

***Index*** Required **Integer**. The node whose editing type is to be set.

***EditingType*** Required [MsoEditingType](#). The editing property of the vertex.

MsoEditingType can be one of these MsoEditingType constants.

**msoEditingAuto**

**msoEditingCorner**

**msoEditingSmooth**

**msoEditingSymmetric**

## Example

This example changes all corner nodes to smooth nodes in shape three on myDocument. Shape three must be a freeform drawing.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(3).Nodes
    For n = 1 to .Count
        If .Item(n).EditingType = msoEditingCorner Then
            .SetEditingType n, msoEditingSmooth
        End If
    Next
End With
```



[Show All](#)

# SetExtrusionDirection Method

Sets the direction that the extrusion's sweep path takes away from the extruded shape.

*expression*.**SetExtrusionDirection**(*PresetExtrusionDirection*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*PresetExtrusionDirection* Required [MsoPresetExtrusionDirection](#). Specifies the extrusion direction.

MsoPresetExtrusionDirection can be one of these MsoPresetExtrusionDirection constants.

**msoExtrusionBottom**

**msoExtrusionBottomLeft**

**msoExtrusionBottomRight**

**msoExtrusionLeft**

**msoExtrusionNone**

**msoExtrusionRight**

**msoExtrusionTop**

**msoExtrusionTopLeft**

**msoExtrusionTopRight**

**msoPresetExtrusionDirectionMixed**

## Remarks

This method sets the [PresetExtrusionDirection](#) property to the direction specified by the *PresetExtrusionDirection* argument.

## Example

This example specifies that the extrusion for shape one on myDocument extend toward the top of the shape and that the lighting for the extrusion come from the left.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(1).ThreeD
    .Visible = True
    .SetExtrusionDirection msoExtrusionTop
    .PresetLightingDirection = msoLightingLeft
End With
```



# SetLinkOnData Method

Sets the name of a procedure that runs whenever a DDE link is updated.

*expression*.**SetLinkOnData**(*Name*, *Procedure*)

*expression* Required. An expression that returns a **Workbook** object.

*Name* Required **String**. The name of the DDE/OLE link, as returned from the **LinkSources** method.

*Procedure* Required **String**. The name of the procedure to be run when the link is updated. This can be either a Microsoft Excel 4.0 macro or a Visual Basic procedure. Set this argument to an empty string ("") to indicate that no procedure should run when the link is updated.

## Example

This example sets the name of the procedure that runs whenever the DDE link is updated.

```
ActiveWorkbook.SetLinkOnData _  
    "WinWord|'C:\MSGFILE.DOC'!DDE_LINK1", _  
    "my_Link_Update_Macro"
```



[Show All](#)

# SetParam Method

Defines a parameter for the specified query table.

*expression*.**SetParam**(*Type*, *Value*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Type** Required [XlParameterType](#).

XlParameterType can be one of these XlParameterType constants.

**xlConstant**. Uses the value specified by the **Value** argument.

**xlPrompt**. Displays a dialog box that prompts the user for the value. The **Value** argument specifies the text shown in the dialog box.

**xlRange**. Uses the value of the cell in the upper-left corner of the range. The **Value** argument specifies a **Range** object

**Value** Required **Variant**. The value of the specified parameter, as shown in the description of the **Type** argument.

## Example

This example changes the SQL statement for query table one. The clause “(city=?)” indicates that the query is a parameter query, and the example sets the value of city to the constant “Oakland.”

```
Set qt = Sheets("sheet1").QueryTables(1)
qt.Sql = "SELECT * FROM authors WHERE (city=?)"
Set param1 = qt.Parameters.Add("City Parameter", _
    xlParamTypeVarChar)
param1.SetParam xlConstant, "Oakland"
qt.Refresh
```

This example sets the value of city to the value of cell A2 on worksheet two.

```
Set qt = Sheets("sheet1").QueryTables(1)
qt.Sql = "SELECT * FROM authors WHERE (city=?)"
Set param1 = qt.Parameters.Add("City Parameter", _
    xlParamTypeVarChar)
param1.SetParam xlRange, Range("sheet2!a1")
qt.Refresh
```



# SetPasswordEncryptionOptions Method

Sets the options for encrypting workbooks using passwords.

*expression*.**SetPasswordEncryptionOptions**(*PasswordEncryptionProvider*,  
*PasswordEncryptionAlgorithm*, *PasswordEncryptionKeyLength*,  
*PasswordEncryptionFileProperties*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

***PasswordEncryptionProvider*** Optional **Variant**. A case sensitive string of the encryption provider.

***PasswordEncryptionAlgorithm*** Optional **Variant**. A case sensitive string of the algorithmic short name (i.e. "RC4").

***PasswordEncryptionKeyLength*** Optional **Variant**. The encryption key length which is a multiple of 8 (40 or greater).

***PasswordEncryptionFileProperties*** Optional **Variant**. **True** (default) to encrypt file properties.

## Remarks

The *PasswordEncryptionProvider*, *PasswordEncryptionAlgorithm*, and *PasswordEncryptionKeyLength* arguments are not independent of each other. A selected encryption provider limits the set of algorithms and key length that can be chosen.

For the *PasswordEncryptionKeyLength* argument there is no inherent limit on the range of the key length. The range is determined by the Cryptographic Service Provider which also determines the cryptographic algorithm.

## Example

This example sets the password encryption options for the active workbook.

```
Sub SetPasswordOptions()
```

```
    ActiveWorkbook.SetPasswordEncryptionOptions _  
        PasswordEncryptionProvider:="Microsoft RSA SChannel Cryptogr  
        PasswordEncryptionAlgorithm:="RC4", _  
        PasswordEncryptionKeyLength:=56, _  
        PasswordEncryptionFileProperties:=True
```

```
End Sub
```



# SetPhonetic Method

Creates [Phonetic](#) objects for all the cells in the specified range.

*expression*.**SetPhonetic**

*expression* An expression that returns a **Range** object.

## Remarks

Any existing **Phonetic** objects in the specified range are automatically overwritten (deleted) by the new objects you add using this method.

## Example

This example creates a **Phonetic** object for each cell in the range A1:A10 on the active worksheet.

```
ActiveSheet.Range("A1:A10").SetPhonetic
```



# SetPosition Method

Sets the location of the node specified by ***Index***. Note that, depending on the editing type of the node, this method may affect the position of adjacent nodes.

*expression*.**SetPosition**(*Index*, *X1*, *Y1*)

*expression* Required. An expression that returns a **ShapeNodes** object.

*Index* Required **Long**. The node whose position is to be set.

*X1* , *Y1* Required **Single**. The position (in points) of the new node relative to the upper-left corner of the document.

## Example

This example moves node two in shape three on myDocument to the right 200 points and down 300 points. Shape three must be a freeform drawing.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(3).Nodes
    pointsArray = .Item(2).Points
    currXvalue = pointsArray(0, 0)
    currYvalue = pointsArray(0, 1)
    .SetPosition 2, currXvalue + 200, currYvalue + 300
End With
```



[Show All](#)

# SetSegmentType Method

Sets the segment type of the segment that follows the node specified by ***Index***. If the node is a control point for a curved segment, this method sets the segment type for that curve. Note that this may affect the total number of nodes by inserting or deleting adjacent nodes.

*expression*.**SetSegmentType**(***Index***, ***SegmentType***)

*expression* Required. An expression that returns one of the objects in the Applies To list.

***Index*** Required **Integer**. The node whose segment type is to be set.

***SegmentType*** Required [MsoSegmentType](#). Specifies if the segment is straight or curved.

MsoSegmentType can be one of these MsoSegmentType constants.

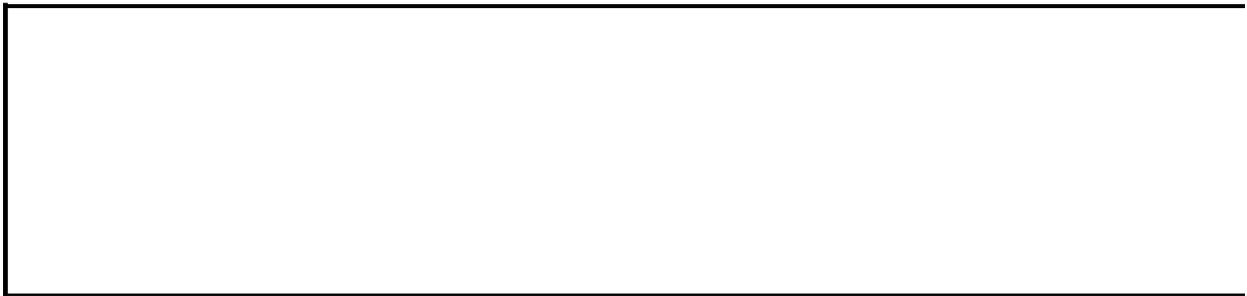
**msoSegmentCurve**

**msoSegmentLine**

## Example

This example changes all straight segments to curved segments in shape three on myDocument. Shape three must be a freeform drawing.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(3).Nodes
    n = 1
    While n <= .Count
        If .Item(n).SegmentType = msoSegmentLine Then
            .SetSegmentType n, msoSegmentCurve
        End If
        n = n + 1
    Wend
End With
```



# SetShapesDefaultProperties Method

Makes the formatting of the specified shape the default formatting for the shape.

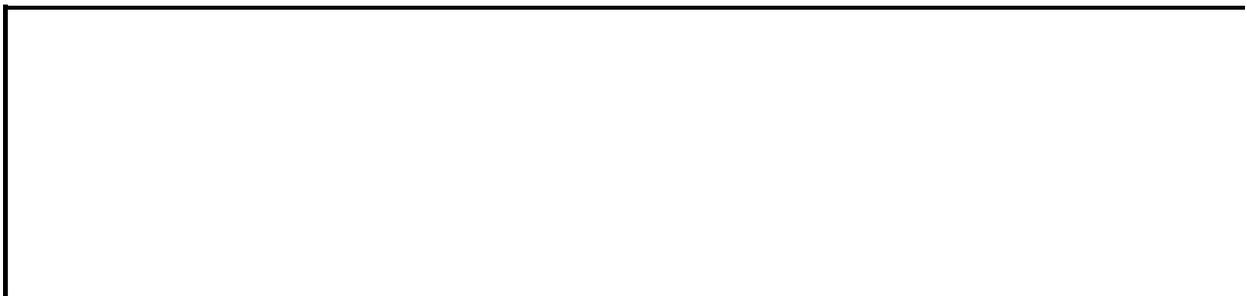
*expression*.**SetShapesDefaultProperties**

*expression* Required. An expression that returns a **Shape** object or **ShapeRange** collection.

## Example

This example adds a rectangle to myDocument, formats the rectangle's fill, sets the rectangle's formatting as the default shape formatting, and then adds another smaller rectangle to the document. The second rectangle has the same fill as the first one.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes
    With .AddShape(msoShapeRectangle, 5, 5, 80, 60)
        With .Fill
            .ForeColor.RGB = RGB(0, 0, 255)
            .BackColor.RGB = RGB(0, 204, 255)
            .Patterned msoPatternHorizontalBrick
        End With
        ' Set formatting as default formatting
        .SetShapesDefaultProperties
    End With
    ' Create new shape with default formatting
    .AddShape msoShapeRectangle, 90, 90, 40, 30
End With
```



# SetSourceData Method

Sets the source data range for the chart.

*expression*.**SetSourceData**(*Source*, *PlotBy*)

*expression* Required. An expression that returns a **Chart** object.

**Source** Required **Range**. The range that contains the source data.

**PlotBy** Optional **Variant**. Specifies the way the data is to be plotted. Can be either of the following **XlRowCol** constants: **xlColumns** or **xlRows**.

## Example

This example sets the source data range for chart one.

```
Charts(1).SetSourceData Source:=Sheets(1).Range("a1:a10"), _  
    PlotBy:=xlColumns
```



[Show All](#)

# SetThreeDFormat Method

Sets the preset extrusion format. Each preset extrusion format contains a set of preset values for the various properties of the extrusion.

*expression*.**SetThreeDFormat**(*PresetThreeDFormat*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*PresetThreeDFormat* Required [MsoPresetThreeDFormat](#). Specifies a preset extrusion format that corresponds to one of the options (numbered from left to right, from top to bottom) displayed when you click the **3-D** button on the **Drawing** toolbar.

MsoPresetThreeDFormat can be one of these MsoPresetThreeDFormat constants.

**msoPresetThreeDFormatMixed**

**msoThreeD1**

**msoThreeD10**

**msoThreeD11**

**msoThreeD12**

**msoThreeD13**

**msoThreeD14**

**msoThreeD15**

**msoThreeD16**

**msoThreeD17**

**msoThreeD18**

**msoThreeD19**

**msoThreeD2**

**msoThreeD20**

**msoThreeD3**

**msoThreeD4**

**msoThreeD5**

**msoThreeD6**

**msoThreeD7**

**msoThreeD8**

**msoThreeD9**

## Remarks

This method sets the [PresetThreeDFormat](#) property to the format specified by the *PresetThreeDFormat* argument.

## Example

This example adds an oval to myDocument and sets its extrusion format to 3D Style 12.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddShape(msoShapeOval, _
    30, 30, 50, 25).ThreeD
    .Visible = True
    .SetThreeDFormat msoThreeD12
End With
```



[Show All](#)

# SetValue Method

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Maps the specified [XPath](#) object to a [ListColumn](#) object or [Range](#) collection. If the **XPath** object has previously been mapped to the **ListColumn** object or **Range** collection, the **SetValue** method sets the properties of the **XPath** object.

*expression.SetValue(Map, XPath, SelectionNamespace, Repeating)*

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Map** Required [XmlMap](#). The schema map that contains information about the **XPath** object.

**XPath** Required **String**. A valid [XPath](#) statement.

**SelectionNamespace** Optional **Variant**. Specifies any [namespace](#) prefixes used in the **XPath** argument. This argument can be omitted if the **XPath** object doesn't contain any prefixes, or if the **XPath** object uses the Microsoft Excel prefixes.

**Repeating** Optional **Boolean**. Specifies whether the **XPath** object is to be bound to a column in an XML list, or mapped to a single cell. Set to **True** to bind the **XPath** object to a column in an XML list.

## Example

The following example creates an XML list based on the "Contacts" schema map that is attached to the workbook, and then uses the **SetValue** method to bind each column to an **XPath** object.

```
Sub CreateXMLList()  
    Dim mapContact As XmlMap  
    Dim strXPath As String  
    Dim lstContacts As ListObject  
    Dim objNewCol As ListColumn  
  
    ' Specify the schema map to use.  
    Set mapContact = ActiveWorkbook.XmlMaps("Contacts")  
  
    ' Create a new list.  
    Set lstContacts = ActiveSheet.ListObjects.Add  
  
    ' Specify the first element to map.  
    strXPath = "/Root/Person/FirstName"  
    ' Map the element.  
    lstContacts.ListColumns(1).XPath.SetValue mapContact, strXPath  
  
    ' Specify the second element to map.  
    strXPath = "/Root/Person/LastName"  
    ' Add a column to the list.  
    Set objNewCol = lstContacts.ListColumns.Add  
    ' Map the element.  
    objNewCol.XPath.SetValue mapContact, strXPath  
  
    strXPath = "/Root/Person/Address/Zip"  
    Set objNewCol = lstContacts.ListColumns.Add  
    objNewCol.XPath.SetValue mapContact, strXPath  
End Sub
```



[Show All](#)

# Show Method

[Show method as it applies to the \*\*Dialog\*\* object.](#)

Displays the built-in dialog box and waits for the user to input data. **Boolean**.

*expression.Show(Arg1, Arg2, Arg3, Arg4, Arg5, Arg6, Arg7, Arg8, Arg9, Arg10, Arg11, Arg12, Arg13, Arg14, Arg15, Arg16, Arg17, Arg18, Arg19, Arg20, Arg21, Arg22, Arg23, Arg24, Arg25, Arg26, Arg27, Arg28, Arg29, Arg30)*

*expression* Required. An expression that returns one of the above objects.

*arg1, arg2, ..., arg30* Optional **Variant**. For built-in dialog boxes only, the initial arguments for the command. For more information, see the "Remarks" section.

*Arg1* Optional **Variant**.

*Arg2* Optional **Variant**.

*Arg3* Optional **Variant**.

*Arg4* Optional **Variant**.

*Arg5* Optional **Variant**.

*Arg6* Optional **Variant**.

*Arg7* Optional **Variant**.

*Arg8* Optional **Variant**.

*Arg9* Optional **Variant**.

*Arg10* Optional **Variant**.

*Arg11* Optional **Variant**.

**Arg12** Optional **Variant**.

**Arg13** Optional **Variant**.

**Arg14** Optional **Variant**.

**Arg15** Optional **Variant**.

**Arg16** Optional **Variant**.

**Arg17** Optional **Variant**.

**Arg18** Optional **Variant**.

**Arg19** Optional **Variant**.

**Arg20** Optional **Variant**.

**Arg21** Optional **Variant**.

**Arg22** Optional **Variant**.

**Arg23** Optional **Variant**.

**Arg24** Optional **Variant**.

**Arg25** Optional **Variant**.

**Arg26** Optional **Variant**.

**Arg27** Optional **Variant**.

**Arg28** Optional **Variant**.

**Arg29** Optional **Variant**.

**Arg30** Optional **Variant**.



[Show method as it applies to the \*\*Range\*\* and \*\*Scenario\*\* objects.](#)

For **Range** objects, scrolls through the contents of the active window to move

the range into view. The range must consist of a single cell in the active document. For **Scenario** objects, shows the scenario by inserting its values on the worksheet. The affected cells are the changing cells of the scenario. **Variant.**

*expression*.**Show**

*expression* Required. An expression that returns one of the above objects.



[Show method as it applies to the CustomView object.](#)

Displays the custom view.

*expression*.**Show**

*expression* Required. An expression that returns one of the above objects.

## Remarks

For built in dialog boxes, this method returns **True** if the user clicks OK, or it returns **False** if the user clicks Cancel.

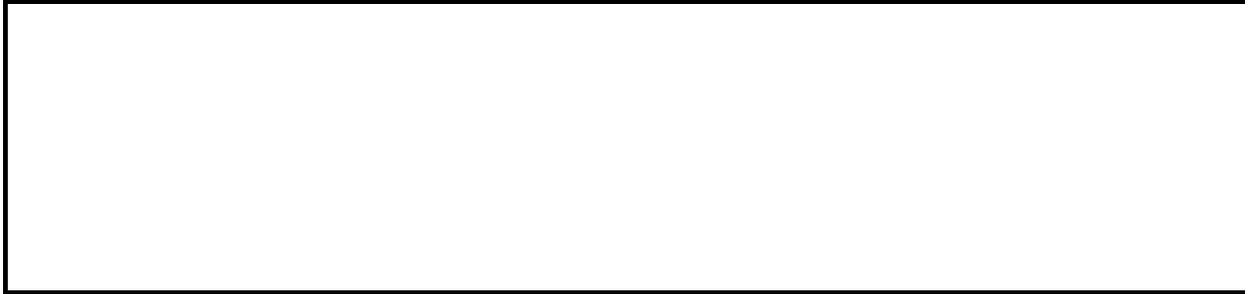
You can use a single dialog box to change many properties at the same time. For example, you can use the **Format Cells** dialog box to change all the properties of the **Font** object.

For some built-in dialog boxes (the **Open** dialog box, for example), you can set initial values using *arg1*, *arg2*, ..., *arg30*. To find the arguments to set, locate the corresponding dialog box constant in [Built-In Dialog Box Argument Lists](#). For example, search for the **xlDialogOpen** constant to find the arguments for the **Open** dialog box. For more information about built-in dialog boxes, see the [Dialogs](#) collection.

## Example

This example displays the **Open** dialog box.

```
Application.Dialogs(xlDialogOpen).Show
```



# ShowAllData Method

Makes all rows of the currently filtered list visible. If AutoFilter is in use, this method changes the arrows to "All."

*expression*.**ShowAllData**

*expression* Required. An expression that returns a **Worksheet** object.

## Example

This example makes all data on Sheet1 visible. The example should be run on a worksheet that contains a list you filtered using the **AutoFilter** command.

```
Worksheets("Sheet1").ShowAllData
```



# ShowDataForm Method

Displays the data form associated with the worksheet.

*expression*.**ShowDataForm**

*expression* Required. An expression that returns a **Worksheet** object.

## Remarks

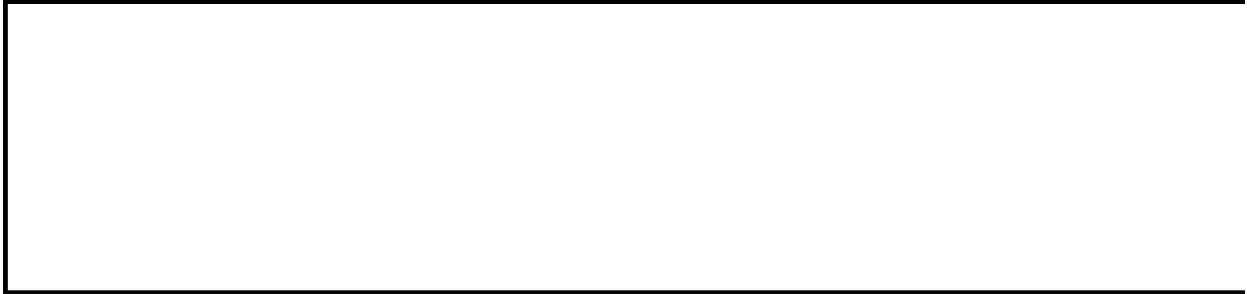
The macro pauses while you're using the data form. When you close the data form, the macro resumes at the line following the **ShowDataForm** method.

This method runs the custom data form, if one exists.

## Example

This example displays the data form for Sheet1.

```
worksheets(1).ShowDataForm
```



# ShowDependents Method

Draws tracer arrows to the direct dependents of the range.

*expression*.**ShowDependents**(*Remove*)

*expression* Required. An expression that returns a **Range** object. Must be a single cell.

**Remove** Optional **Variant**. **True** to remove one level of tracer arrows to direct dependents. **False** to expand one level of tracer arrows. The default value is **False**.

## Example

This example draws tracer arrows to dependents of the active cell on Sheet1.

```
Worksheets("Sheet1").Activate  
ActiveCell.ShowDependents
```

This example removes the tracer arrow for one level of dependents of the active cell on Sheet1.

```
Worksheets("Sheet1").Activate  
ActiveCell.ShowDependents Remove:=True
```



# ShowErrors Method

Draws tracer arrows through the precedents tree to the cell that's the source of the error, and returns the range that contains that cell.

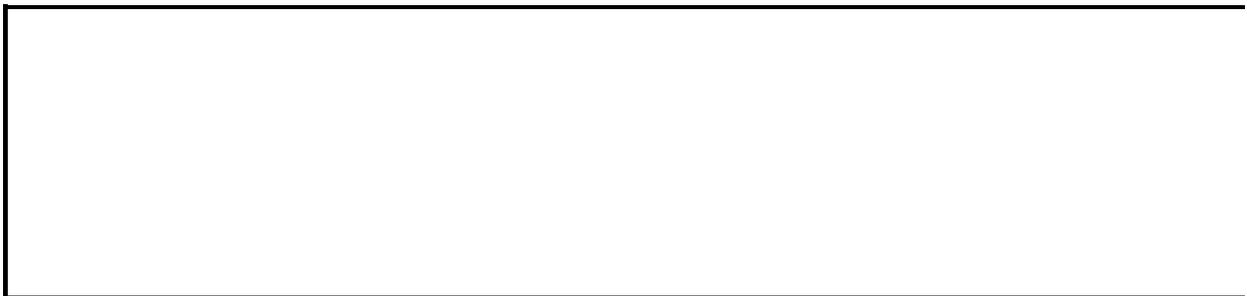
*expression*.**ShowErrors**

*expression* Required. An expression that returns a **Range** object.

## Example

This example displays a red tracer arrow if there's an error in the active cell on Sheet1.

```
Worksheets("Sheet1").Activate  
If IsError(ActiveCell.Value) Then  
    ActiveCell.ShowErrors  
End If
```



# ShowLevels Method

Displays the specified number of row and/or column levels of an outline.

*expression*.**ShowLevels**(*RowLevels*, *ColumnLevels*)

*expression* Required. An expression that returns an **Outline** object.

**RowLevels** Optional **Variant**. Specifies the number of row levels of an outline to display. If the outline has fewer levels than the number specified, Microsoft Excel displays all the levels. If this argument is 0 (zero) or is omitted, no action is taken on rows.

**ColumnLevels** Optional **Variant**. Specifies the number of column levels of an outline to display. If the outline has fewer levels than the number specified, Microsoft Excel displays all the levels. If this argument is 0 (zero) or is omitted, no action is taken on columns.

## Remarks

You must specify at least one argument.

## Example

This example displays row levels one through three and column level one of the outline on Sheet1.

```
Worksheets("Sheet1").Outline _  
    .ShowLevels rowLevels:=3, columnLevels:=1
```



[Show All](#)

# ShowPages Method

Creates a new PivotTable report for each item in the page field. Each new report is created on a new worksheet.

*expression*.**ShowPages**(*PageField*)

*expression* Required. An expression that returns a **PivotTable** object.

**PageField** Optional **Variant**. A string that names a single page field in the report.

## Remarks

This method isn't available for [OLAP](#) data sources.

## Example

This example creates a new PivotTable report for each item in the page field, which is the field named "Country."

```
Set pvtTable = Worksheets("Sheet1").Range("A3").PivotTable  
pvtTable.ShowPages "Country"
```



# ShowPrecedents Method

Draws tracer arrows to the direct precedents of the range.

*expression*.ShowPrecedents(*Remove*)

*expression* Required. An expression that returns a **Range** object. Must be a single cell.

**Remove** Optional **Variant**. **True** to remove one level of tracer arrows to direct precedents. **False** to expand one level of tracer arrows. The default value is **False**.

## Example

This example draws tracer arrows to the precedents of the active cell on Sheet1.

```
Worksheets("Sheet1").Activate  
ActiveCell.ShowPrecedents
```

This example removes the tracer arrow for one level of precedents of the active cell on Sheet1.

```
Worksheets("Sheet1").Activate  
ActiveCell.ShowPrecedents remove:=True
```



# SmallScroll Method

Scrolls the contents of the window by rows or columns.

*expression*.**SmallScroll**(*Down, Up, ToRight,ToLeft*)

*expression* Required. An expression that returns a **Window** object.

**Down** Optional **Variant**. The number of rows to scroll the contents down.

**Up** Optional **Variant**. The number of rows to scroll the contents up.

**ToRight** Optional **Variant**. The number of columns to scroll the contents to the right.

**ToLeft** Optional **Variant**. The number of columns to scroll the contents to the left.

## Remarks

If ***Down*** and ***Up*** are both specified, the contents of the window are scrolled by the difference of the arguments. For example, if ***Down*** is 3 and ***Up*** is 6, the the contents are scrolled up three rows.

If ***ToLeft*** and ***ToRight*** are both specified, the contents of the window are scrolled by the difference of the arguments. For example, if ***ToLeft*** is 3 and ***ToRight*** is 6, the contents are scrolled to the right three columns.

Any of these arguments can be a negative number.

## Example

This example scrolls the contents of the active window of Sheet1 down three rows.

```
Worksheets("Sheet1").Activate  
Activewindow.SmallScroll down:=3
```



# Solid Method

Sets the specified fill to a uniform color. Use this method to convert a gradient, textured, patterned, or background fill back to a solid fill.

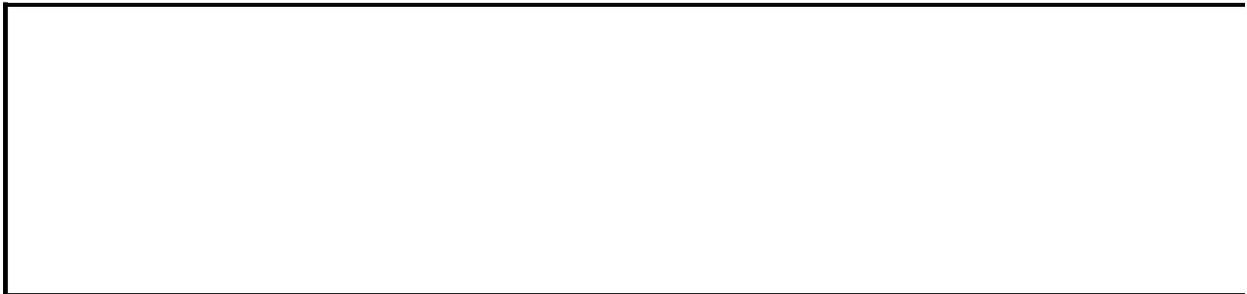
*expression*.**Solid**

*expression* Required. An expression that returns a **FillFormat** object.

## Example

This example converts all fills on myDocument to uniform red fills.

```
Set myDocument = Worksheets(1)
For Each s In myDocument.Shapes
  With s.Fill
    .Solid
    .ForeColor.RGB = RGB(255, 0, 0)
  End With
Next
```



[Show All](#)

# Sort Method

Sorts a PivotTable report, a range, or the active region if the specified range contains only one cell.

*expression.Sort(Key1, Order1, Key2, Type, Order2, Key3, Order3, Header, OrderCustom, MatchCase, Orientation, SortMethod, DataOption1, DataOption2, DataOption3)*

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Key1** Optional **Variant**. The first sort field, as either text (a PivotTable field or range name) or a **Range** object ("Dept" or Cells(1, 1), for example).

**Order1** Optional [XlSortOrder](#). The sort order for the field or range specified in **Key1**.

XlSortOrder can be one of these XlSortOrder constants.

**xlDescending**. Sorts **Key1** in descending order.

**xlAscending** *default*. Sorts **Key1** in ascending order.

**Key2** Optional **Variant**. The second sort field, as either text (a PivotTable field or range name) or a **Range** object. If you omit this argument, there's no second sort field. Cannot be used when sorting Pivot Table reports.

**Type** Optional **Variant**. Specifies which elements are to be sorted. Use this argument only when sorting PivotTable reports.

XlSortType can be one of these XlSortType constants.

**xlSortLabels**. Sorts the PivotTable report by labels.

**xlSortValues**. Sorts the PivotTable report by values.

**Order2** Optional [XlSortOrder](#). The sort order for the field or range specified in **Key2**. Cannot be used when sorting PivotTable reports.

XlSortOrder can be one of these XlSortOrder constants.

**xlDescending**. Sorts **Key2** in descending order.

**xlAscending** *default*. Sorts **Key2** in ascending order.

**Key3** Optional **VARIANT**. The third sort field, as either text (a range name) or a **Range** object. If you omit this argument, there's no third sort field. Cannot be used when sorting PivotTable reports.

**Order3** Optional [XlSortOrder](#). The sort order for the field or range specified in **Key3**. Cannot be used when sorting PivotTable reports.

XlSortOrder can be one of these XlSortOrder constants.

**xlDescending**. Sorts **Key3** in descending order.

**xlAscending** *default*. Sorts **Key3** in ascending order.

**Header** Optional [XlYesNoGuess](#). Specifies whether or not the first row contains headers. Cannot be used when sorting PivotTable reports.

XlYesNoGuess can be one of these XlYesNoGuess constants.

**xlGuess**. Let Microsoft Excel determine whether there's a header, and to determine where it is, if there is one.

**xlNo** *default*. (The entire range should be sorted).

**xlYes**. (The entire range should not be sorted).

**OrderCustom** Optional **VARIANT**. This argument is a one-based integer offset to the list of custom sort orders. If you omit **OrderCustom**, a normal sort is used.

**MatchCase** Optional **VARIANT**. **True** to do a case-sensitive sort; **False** to do a sort that's not case sensitive. Cannot be used when sorting PivotTable reports.

**Orientation** Optional [XlSortOrientation](#). The sort orientation.

XlSortOrientation can be one of these XlSortOrientation constants.

**xlSortRows** *default*. Sorts by row.

**xlSortColumns**. Sorts by column.

**SortMethod** Optional [XlSortMethod](#). The type of sort. Some of these

constants may not be available to you, depending on the language support (U.S. English, for example) that you've selected or installed.

XLSortMethod can be one of these XLSortMethod constants.

**xlStroke** Sorting by the quantity of strokes in each character.

**xlPinYin** *default*. Phonetic Chinese sort order for characters.

**DataOption1** Optional [XLSortDataOption](#). Specifies how to sort text in key 1. Cannot be used when sorting PivotTable reports.

XLSortDataOption can be one of these XLSortDataOption constants.

**xlSortTextAsNumbers**. Treat text as numeric data for the sort.

**xlSortNormal** *default*. Sorts numeric and text data separately.

**DataOption2** Optional [XLSortDataOption](#). Specifies how to sort text in key 2. Cannot be used when sorting PivotTable reports.

XLSortDataOption can be one of these XLSortDataOption constants.

**xlSortTextAsNumbers**. Treats text as numeric data for the sort.

**xlSortNormal** *default*. Sorts numeric and text data separately.

**DataOption3** Optional [XLSortDataOption](#). Specifies how to sort text in key 3. Cannot be used when sorting PivotTable reports.

XLSortDataOption can be one of these XLSortDataOption constants.

**xlSortTextAsNumbers**. Treats text as numeric data for the sort.

**xlSortNormal** *default*. Sorts numeric and text data separately.

## Remarks

The settings for ***Header***, ***Order1***, ***Order2***, ***Order3***, ***OrderCustom***, and ***Orientation*** are saved, for the particular worksheet, each time you use this method. If you don't specify values for these arguments the next time you call the method, the saved values are used. Set these arguments explicitly each time you use **Sort** method, if you choose not to use the saved values.

Text strings which are not convertible to numeric data are sorted normally.

**Note** If no arguments are defined with the **Sort** method, Microsoft Excel will sort the selection, chosen to be sorted, in ascending order.

## Example

This example sorts the range A1:C20 on Sheet1, using cell A1 as the first sort key and cell B1 as the second sort key. The sort is done in ascending order by row, and there are no headers. This example assumes there is data in the range A1:C20.

```
Sub SortRange1()  
    Worksheets("Sheet1").Range("A1:C20").Sort _  
        Key1:=Worksheets("Sheet1").Range("A1"), _  
        Key2:=Worksheets("Sheet1").Range("B1")  
End Sub
```

This example sorts the region that contains cell A1 (the active region) on Sheet1, sorting by the data in the first column and automatically using a header row if one exists. This example assumes there is data in the active region, which includes cell A1. The **Sort** method determines the active region automatically.

```
Sub SortRange2()  
    Worksheets("Sheet1").Range("A1").Sort _  
        Key1:=Worksheets("Sheet1").Columns("A"), _  
        Header:=xlGuess  
End Sub
```



[Show All](#)

# SortSpecial Method

Uses East Asian sorting methods to sort the range, a PivotTable report, or uses the method for the active region if the range contains only one cell. For example, Japanese sorts in the order of the Kana syllabary. For more information, see the argument list.

*expression*.**SortSpecial**(*SortMethod*, *Key1*, *Order1*, *Type*, *Key2*, *Order2*, *Key3*, *Order3*, *Header*, *OrderCustom*, *MatchCase*, *Orientation*, *DataOption1*, *DataOption2*, *DataOption3*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**SortMethod** Optional [XISortMethod](#). The type of sort. Some of these constants may not be available to you, depending on the language support (U.S. English, for example) that you've selected or installed.

XISortMethod can be one of these XISortMethod constants.

**xlStroke**. Sorting by the quantity of strokes in each character.

**xlPinYin** *default*. Phonetic Chinese sort order for characters.

**Key1** Optional **Variant**. The first sort field, as either text (a PivotTable field or range name) or a **Range** object ("Dept" or Cells(1, 1), for example).

**Order1** Optional [XISortOrder](#). The sort order for the field or range specified in the **Key1** argument.

XISortOrder can be one of these XISortOrder constants.

**xlDescending**. Sorts **Key1** in descending order.

**xlAscending** *default*. Sorts **Key1** in ascending order.

**Type** Optional **Variant**. Specifies which elements are to be sorted. Use this argument only when sorting PivotTable reports.

**Key2** Optional **Variant**. The second sort field, as either text (a PivotTable field

or range name) or a **Range** object. If you omit this argument, there's no second sort field. Cannot be used when sorting PivotTable reports.

XISortType can be one of these XISortType constants.

**xlSortLabels**. Sorts the PivotTable report by labels.

**xlSortValues**. Sorts the PivotTable report by values.

**Order2** Optional **XISortOrder**. The sort order for the field or range specified in the **Key2** argument. Cannot be used when sorting PivotTable reports.

XISortOrder can be one of these XISortOrder constants.

**xlDescending**. Sorts **Key2** in descending order.

**xlAscending** *default*. Sorts **Key2** in ascending order.

**Key3** Optional **Variant**. The third sort field, as either text (a range name) or a **Range** object. If you omit this argument, there's no third sort field. Cannot be used when sorting PivotTable reports.

**Order3** Optional **XISortOrder**. The sort order for the field or range specified in the **Key3** argument. Cannot be used when sorting PivotTable reports.

XISortOrder can be one of these XISortOrder constants.

**xlDescending**. Sorts **Key3** in descending order.

**xlAscending** *default*. Sorts **Key3** in ascending order.

**Header** Optional **XIYesNoGuess**. Specifies whether or not the first row contains headers. Cannot be used when sorting PivotTable reports.

XIYesNoGuess can be one of these XIYesNoGuess constants.

**xlGuess**. Lets Microsoft Excel determine whether there's a header, and to determine where it is, if there is one.

**xlNo** *default*. The entire range should be sorted.

**xlYes**. The entire range should not be sorted.

**OrderCustom** Optional **Variant**. This argument is a one-based integer offset to the list of custom sort orders. If you omit **OrderCustom**, (normal sort order) is used.

**MatchCase** Optional **Variant**. **True** to do a case-sensitive sort; **False** to do a sort that's not case sensitive. Cannot be used when sorting PivotTable reports.

**Orientation** Optional [XISortOrientation](#). The sort orientation.

XISortOrientation can be one of these XISortOrientation constants.

**xlSortRows** *default*. The sort is done by row.

**xlSortColumns**. The sort is done by column.

**DataOption1** Optional [XISortDataOption](#). Specifies how to sort text in key1. Cannot be used when sorting PivotTable reports.

XISortDataOption can be one of these XISortDataOption constants.

**xlSortTextAsNumbers**. Treats text as numeric data for the sort.

**xlSortNormal** *default*. Sorts numeric and text data separately.

**DataOption2** Optional [XISortDataOption](#). Specifies how to sort text in key 2. Cannot be used when sorting PivotTable reports.

XISortDataOption can be one of these XISortDataOption constants.

**xlSortTextAsNumbers**. Treats text as numeric data for the sort.

**xlSortNormal** *default*. Sorts numeric and text data separately.

**DataOption3** Optional [XISortDataOption](#). Specifies how to sort text in key 3. Cannot be used when sorting PivotTable reports.

XISortDataOption can be one of these XISortDataOption constants.

**xlSortTextAsNumbers**. Treats text numeric data for the sort.

**xlSortNormal** *default*. Sorts numeric and text data separately.

## Remarks

**Note** : If no arguments are defined with the **Sort** method, Microsoft Excel will sort the selection, chosen to be sorted, in ascending order.

## Example

This example sorts the range A1:A5 using Pin Yin (phonetic Chinese sort order for characters). In order to sort Chinese characters, this example assumes the user has Chinese language support for Microsoft Excel. Even without Chinese language support, Excel will default to sorting any numbers placed within the specified range for this example. This example assumes there is data contained in the range A1:A5.

```
Sub SpecialSort()
```

```
    Application.Range("A1:A5").SortSpecial SortMethod:=xlPinYin
```

```
End Sub
```



[Show All](#)

# Speak Method

 [Speak method as it applies to the \*\*Range\*\* object.](#)

Causes the cells of the range to be spoken in row order or column order.

*expression*.**Speak**(*SpeakDirection*, *SpeakFormulas*)

*expression* Required. An expression that returns a **Range** object.

**SpeakDirection** Optional **Variant**. The speak direction, by rows or columns.

**SpeakFormulas** Optional **Variant**. **True** will cause formulas to be sent to the Text-To-Speech (TTS) engine for cells that have formulas. The value is sent if the cells do not have formulas. **False** (default) will cause values to always be sent to the TTS engine.

 [Speak method as it applies to the \*\*Speech\*\* object.](#)

Microsoft Excel plays back the text string that is passed as an argument.

*expression*.**Speak**(*Text*, *SpeakAsync*, *SpeakXML*, *Purge*)

*expression* Required. An expression that returns a **Speech** object.

**Text** Required **String**. The text to be spoken.

**SpeakAsync** Optional **Variant**. **True** will cause the **Text** to be spoken asynchronously (the method will not wait of the Text to be spoken). **False** will cause the **Text** to be spoken synchronously (the method waits for the **Text** to be spoken before continuing). The default is **False**.

**SpeakXML** Optional **Boolean**. **True** will cause the **Text** to be interpreted as XML. **False** will cause the **Text** to not be interpreted as XML, so any XML tags will be read and not interpreted. The default is **False**.

**Purge** Optional **Variant**. **True** will cause current speech to be terminated and

any buffered text to be purged before ***Text*** is spoken. **False** will not cause the current speech to be terminated and will not purge the buffered text before ***Text*** is spoken. The default is **False**.

## Example

In this example, Microsoft Excel speaks "Hello".

```
Sub UseSpeech()
```

```
    Application.Speech.Speak "Hello"
```

```
End Sub
```



[Show All](#)

# SpecialCells Method

Returns a [Range](#) object that represents all the cells that match the specified type and value. **Range** object.

*expression*.**SpecialCells**(*Type*, *Value*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*Type* Required [xlCellType](#). The cells to include.

XlCellType can be one of these XlCellType constants.

**xlCellTypeAllFormatConditions**. Cells of any format

**xlCellTypeAllValidation**. Cells having validation criteria

**xlCellTypeBlanks**. Empty cells

**xlCellTypeComments**. Cells containing notes

**xlCellTypeConstants**. Cells containing constants

**xlCellTypeFormulas**. Cells containing formulas

**xlCellTypeLastCell**. The last cell in the used range

**xlCellTypeSameFormatConditions**. Cells having the same format

**xlCellTypeSameValidation**. Cells having the same validation criteria

**xlCellTypeVisible**. All visible cells

*Value* Optional **Variant**. If *Type* is either **xlCellTypeConstants** or **xlCellTypeFormulas**, this argument is used to determine which types of cells to include in the result. These values can be added together to return more than one type. The default is to select all constants or formulas, no matter what the type. Can be one of the following [XlSpecialCellsValue](#) constants:

XlSpecialCellsValue can be one of these XlSpecialCellsValue constants.

**xlErrors**

**xlLogical**

**xlNumbers**

**xlTextValues**

## Example

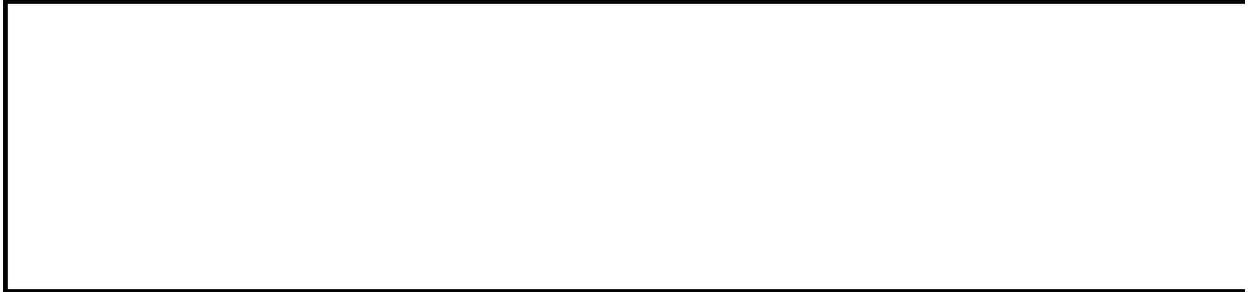
This example selects the last cell in the used range of Sheet1.

```
Worksheets("Sheet1").Activate  
ActiveSheet.Cells.SpecialCells(xlCellTypeLastCell).Activate
```



# SubscribeTo Method

You have requested Help for a Visual Basic keyword used only on the Macintosh. For information about this keyword, consult the language reference Help included with Microsoft Office Macintosh Edition.



[Show All](#)

# Subtotal Method

 [Subtotal method as it applies to the \*\*WorksheetFunction\*\* object.](#)

Creates subtotals. For information about using the **Subtotal** worksheet function in Visual Basic, see [Using Worksheet Functions in Visual Basic](#).

*expression*.**Subtotal**(*Arg1*, *Arg2*, *Arg3*, *Arg4*, *Arg5*, *Arg6*, *Arg7*, *Arg8*, *Arg9*, *Arg10*, *Arg11*, *Arg12*, *Arg13*, *Arg14*, *Arg15*, *Arg16*, *Arg17*, *Arg18*, *Arg19*, *Arg20*, *Arg21*, *Arg22*, *Arg23*, *Arg24*, *Arg25*, *Arg26*, *Arg27*, *Arg28*, *Arg29*, *Arg30*)

*expression* Required. An expression that returns a **WorksheetFunction** object.

**Arg1** Required **Double**.

**Arg2** Required **Range** object.

**Arg3-Arg30** Optional **Variant**.

 [Subtotal method as it applies to the \*\*Range\*\* object.](#)

Creates subtotals for the range (or the current region, if the range is a single cell).

For information about using the **Subtotal** worksheet function in Visual Basic, see [Using Worksheet Functions in Visual Basic](#).

*expression*.**Subtotal**(*GroupBy*, *Function*, *TotalList*, *Replace*, *PageBreaks*, *SummaryBelowData*)

*expression* Required. An expression that returns a **Range** object.

**GroupBy** Required **Long**. The field to group by, as a one-based integer offset. For more information, see the example.

**Function** Required [XlConsolidationFunction](#). The subtotal function.

XlConsolidationFunction can be one of these XlConsolidationFunction constants.

**xlAverage**

**xlCount**

**xlCountNums**

**xlMax**

**xlMin**

**xlProduct**

**xlStDev**

**xlStDevP**

**xlSum**

**xlUnknown**

**xlVar**

**xlVarP**

**TotalList** Required **Variant**. An array of 1-based field offsets, indicating the fields to which the subtotals are added. For more information, see the example.

**Replace** Optional **Variant**. **True** to replace existing subtotals. The default value is **False**.

**PageBreaks** Optional **Variant**. **True** to add page breaks after each group. The default value is **False**.

**SummaryBelowData** Optional [XlSummaryRow](#). Places the summary data relative to the subtotal.

XlSummaryRow can be one of these XlSummaryRow constants.

**xlSummaryAbove**

**xlSummaryBelow** *default*

## Example

This example creates subtotals for the selection on Sheet1. The subtotals are sums grouped by each change in field one, with the subtotals added to fields two and three.

```
Worksheets("Sheet1").Activate  
Selection.Subtotal GroupBy:=1, Function:=xlSum, _  
    TotalList:=Array(2, 3)
```



# SwapNode Method

Swaps the source diagram node with a target diagram node.

*expression*.**SwapNode**(*pTargetNode*, *swapChildren*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*pTargetNode* Required **DiagramNode** object. The target diagram node to be replaced.

*swapChildren* Optional **Boolean**. The child nodes of the target and source nodes being swapped. Any child diagram nodes are moved along with their corresponding root nodes. Default is **True**, which swaps the child nodes.

## Example

The following example swaps the second diagram node of a newly-created diagram with the last node.

```
Sub SwapNode()  
  
    Dim nodRoot As DiagramNode  
    Dim nodPrev As DiagramNode  
    Dim shDiagram As Shape  
    Dim intCount As Integer  
  
    Set shDiagram = ActiveSheet.Shapes.AddDiagram _  
        (Type:=msoDiagramRadial, Left:=10, Top:=15, _  
         Width:=400, Height:=475)  
    Set nodRoot = shDiagram.DiagramNode.Children.AddNode  
  
    ' Add 3 child nodes to the root node.  
    For intCount = 1 To 3  
        nodRoot.Children.AddNode  
    Next  
  
    ' Swap the second node with the fourth node.  
    nodRoot.Children.Item(2).SwapNode _  
        pTargetNode:=nodRoot.Diagram.Nodes(4), _  
        swapChildren:=True  
  
End Sub
```



# Table Method

Creates a data table based on input values and formulas that you define on a worksheet.

*expression*.**Table**(**RowInput**, **ColumnInput**)

*expression* Required. An expression that returns an object in the Applies To list.

**RowInput** Optional **Variant**. A single cell to use as the row input for your table.

**ColumnInput** Optional **Variant**. A single cell to use as the column input for your table.

## Remarks

Use data tables to perform a what-if analysis by changing certain constant values on your worksheet to see how values in other cells are affected.

## Example

This example creates a formatted multiplication table in cells A1:K11 on Sheet1.

```
Set dataTableRange = Worksheets("Sheet1").Range("A1:K11")
Set rowInputCell = Worksheets("Sheet1").Range("A12")
Set columnInputCell = Worksheets("Sheet1").Range("A13")
```

```
Worksheets("Sheet1").Range("A1").Formula = "=A12*A13"
For i = 2 To 11
    Worksheets("Sheet1").Cells(i, 1) = i - 1
    Worksheets("Sheet1").Cells(1, i) = i - 1
Next i
dataTableRange.Table rowInputCell, columnInputCell
With Worksheets("Sheet1").Range("A1").CurrentRegion
    .Rows(1).Font.Bold = True
    .Columns(1).Font.Bold = True
    .Columns.AutoFit
End With
```



[Show All](#)

# Text Method

 [Text method as it applies to the \*\*WorksheetFunction\*\* object.](#)

Converts a value to text in a specific number format.

*expression*.**Text**(*Arg1*, *Arg2*)

*expression* Required. An expression that returns one of the above objects.

**Arg1** Required **Variant**. A numeric value, a formula that evaluates to a numeric value, or a reference to a cell containing a numeric value.

**Arg2** Required **String**. A number format in text form in the **Category** box on the **Number** tab in the **Format Cells** dialog box.

 [Text method as it applies to the \*\*Comment\*\* object.](#)

Sets comment text.

*expression*.**Text**(*Text*, *Start*, *Overwrite*)

*expression* Required. An expression that returns one of the above objects.

**Text** Optional **Variant**. The text to be added.

**Start** Optional **Variant**. The character number where the added text will be placed. If this argument is omitted, any existing text in the comment is deleted.

**Overwrite** Optional **Variant**. **True** to overwrite the existing text. The default value is **False** (text is inserted).

## Example

This example adds a comment to cell E5 on sheet one.

```
With Worksheets(1).Range("e5").AddComment  
    .Visible = False  
    .Text "reviewed on " & Date  
End With
```



[Show All](#)

# TextToColumns Method

Parses a column of cells that contain text into several columns.

*expression*.**TextToColumns**(*Destination*, *DataType*, *TextQualifier*, *ConsecutiveDelimiter*, *Tab*, *Semicolon*, *Comma*, *Space*, *Other*, *OtherChar*, *FieldInfo*, *DecimalSeparator*, *ThousandsSeparator*, *TrailingMinusNumbers*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Destination** Optional **Variant**. A **Range** object that specifies where Microsoft Excel will place the results. If the range is larger than a single cell, the top left cell is used.

**DataType** Optional [XLTextParsingType](#). The format of the text to be split into columns.

XLTextParsingType can be one of these XLTextParsingType constants.

**xIDelimited** *default*

**xIFixedWidth**

**TextQualifier** Optional [XLTextQualifier](#).

XLTextQualifier can be one of these XLTextQualifier constants.

**xlTextQualifierDoubleQuote** *default*

**xlTextQualifierNone**

**xlTextQualifierSingleQuote**

**ConsecutiveDelimiter** Optional **Variant**. **True** to have Microsoft Excel consider consecutive delimiters as one delimiter. The default value is **False**.

**Tab** Optional **Variant**. **True** to have **DataType** be **xIDelimited** and to have the tab character be a delimiter. The default value is **False**.

**Semicolon** Optional **Variant**. **True** to have **DataType** be **xIDelimited** and to

have the semicolon be a delimiter. The default value is **False**.

**Comma** Optional **Variant**. **True** to have **DataType** be **xlDelimited** and to have the comma be a delimiter. The default value is **False**.

**Space** Optional **Variant**. **True** to have **DataType** be **xlDelimited** and to have the space character be a delimiter. The default value is **False**.

**Other** Optional **Variant**. **True** to have **DataType** be **xlDelimited** and to have the character specified by the **OtherChar** argument be a delimiter. The default value is **False**.

**OtherChar** Optional **Variant** (required if **Other** is **True**). The delimiter character when **Other** is **True**. If more than one character is specified, only the first character of the string is used; the remaining characters are ignored.

**FieldInfo** Optional **Variant**. An array containing parse information for the individual columns of data. The interpretation depends on the value of **DataType**. When the data is delimited, this argument is an array of two-element arrays, with each two-element array specifying the conversion options for a particular column. The first element is the column number (1-based), and the second element is one of the [xlColumnDataType](#) constants specifying how the column is parsed.

XlColumnDataType can be one of these XlColumnDataType constants.

**xlGeneralFormat**. General

**xlTextFormat**. Text

**xlMDYFormat**. MDY Date

**xlDMYFormat**. DMY Date

**xlYMDFormat**. YMD Date

**xlMYDFormat**. MYD Date

**xlDYMFormat**. DYM Date

**xlYDMFormat**. YDM Date

**xlEMDFormat.** EMD Date

**xlSkipColumn.** Skip Column

You can use **xlEMDFormat** only if Taiwanese language support is installed and selected. The **xlEMDFormat** constant specifies that Taiwanese era dates are being used.

The column specifiers can be in any order. If a given column specifier is not present for a particular column in the input data, the column is parsed with the **General** setting. This example causes the third column to be skipped, the first column to be parsed as text, and the remaining columns in the source data to be parsed with the **General** setting.

```
Array(Array(3, 9), Array(1, 2))
```

If the source data has fixed-width columns, the first element of each two-element array specifies the starting character position in the column (as an integer; 0 (zero) is the first character). The second element of the two-element array specifies the parse option for the column as a number from 1 through 9, as listed above.

The following example parses two columns from a fixed-width file, with the first column starting at the beginning of the line and extending for 10 characters. The second column starts at position 15 and goes to the end of the line. To avoid including the characters between position 10 and position 15, Microsoft Excel adds a skipped column entry.

```
Array(Array(0, 1), Array(10, 9), Array(15, 1))
```

**DecimalSeparator** Optional **String**. The decimal separator that Microsoft Excel uses when recognizing numbers. The default setting is the system setting.

**ThousandsSeparator** Optional **String**. The thousands separator that Excel uses when recognizing numbers. The default setting is the system setting.

**TrailingMinusNumbers** Optional **VARIANT**. Numbers that begin with a minus character.

The following table shows the results of importing text into Excel for various import settings. Numeric results are displayed in the rightmost column.

<b>System decimal separator</b>	<b>System thousands separator</b>	<b>Decimal separator value</b>	<b>Thousands separator value</b>	<b>Original text</b>	<b>Cell value (data type)</b>
Period	Comma	Comma	Period	123.123,45	123,123.45 (numeric)
Period	Comma	Comma	Comma	123.123,45	123.123,45 (text)
Comma	Period	Comma	Period	123,123.45	123,123.45 (numeric)
Period	Comma	Period	Comma	123 123.45	123 123.45 (text)
Period	Comma	Period	Space	123 123.45	123,123.45 (numeric)

## Example

This example converts the contents of the Clipboard, which contains a space-delimited text table, into separate columns on Sheet1. You can create a simple space-delimited table in Notepad or WordPad (or another text editor), copy the text table to the Clipboard, switch to Microsoft Excel, and then run this example.

```
Worksheets("Sheet1").Activate  
ActiveSheet.Paste  
Selection.TextToColumns DataType:=xlDelimited, _  
    ConsecutiveDelimiter:=True, Space:=True
```



# ToggleVerticalText Method

Switches the text flow in the specified WordArt from horizontal to vertical, or vice versa.

*expression*.**ToggleVerticalText**

*expression* Required. An expression that returns a **TextEffectFormat** object.

## Remarks

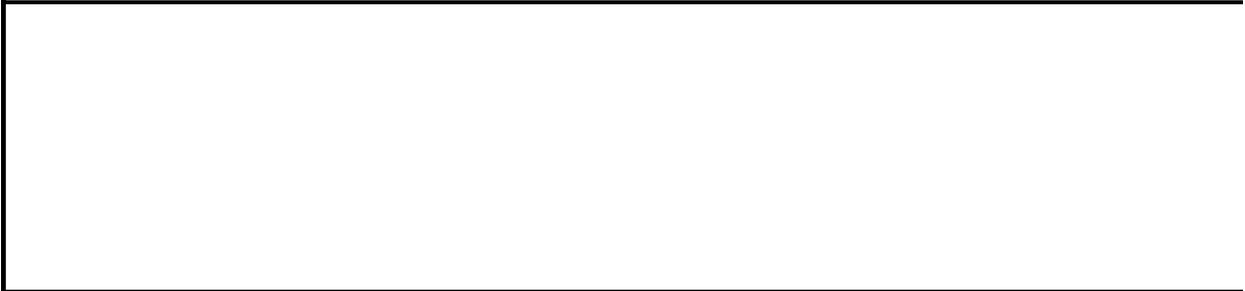
Using the **ToggleVerticalText** method swaps the values of the **Width** and **Height** properties of the **Shape** object that represents the WordArt and leaves the **Left** and **Top** properties unchanged.

The **Flip** method and **Rotation** property of the **Shape** object and the **RotatedChars** property and **ToggleVerticalText** method of the **TextEffectFormat** object all affect the character orientation and the direction of text flow in a **Shape** object that represents WordArt. You may have to experiment to find out how to combine the effects of these properties and methods to get the result you want.

## Example

This example adds WordArt that contains the text "Test" to myDocument and switches from horizontal text flow (the default for the specified WordArt style, **msoTextEffect1**) to vertical text flow.

```
Set myDocument = Worksheets(1)
Set newWordArt = myDocument.Shapes.AddTextEffect( _
    PresetTextEffect:=msoTextEffect1, Text:="Test", _
    FontName:="Arial Black", FontSize:=36, _
    FontBold:=False, FontItalic:=False, Left:=100, _
    Top:=100)
newWordArt.TextEffect.ToggleVerticalText
```



# TransferChildren Method

Transfers the child nodes of a source diagram node to a receiving diagram node.

*expression*.**TransferChildren**(*pReceivingNode*)

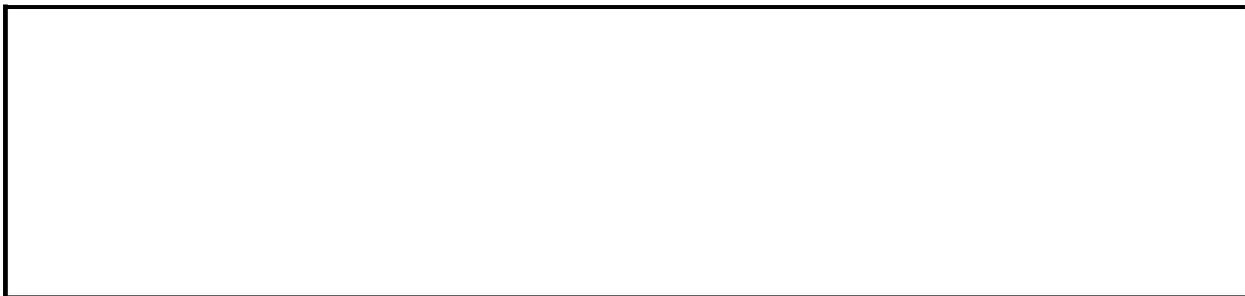
*expression* Required. An expression that returns one of the objects in the Applies To list.

***pReceivingNode*** Required **DiagramNode** object. The diagram node receiving the child nodes being transferred from the node signified in *expression* .

## Example

The following example transfers the child nodes of a newly-created diagram from one node to another.

```
Sub TransferChildNodes()  
    Dim dgnNode As DiagramNode  
    Dim shpDiagram As Shape  
    Dim intCount As Integer  
  
    'Add organizational chart to current document  
    Set shpDiagram = ActiveSheet.Shapes.AddDiagram _  
        (Type:=msoDiagramOrgChart, Left:=10, _  
        Top:=15, Width:=400, Height:=475)  
  
    'Add first node to organizational chart  
    Set dgnNode = shpDiagram.DiagramNode.Children.AddNode  
  
    'Add three child nodes to first node  
    For intCount = 1 To 3  
        dgnNode.Children.AddNode  
    Next intCount  
  
    'Add three child nodes to the first child node  
    'of the first node  
    For intCount = 1 To 3  
        dgnNode.Children.Item(1).Children.AddNode  
    Next intCount  
  
    'Moves the child nodes of the first child node  
    'so they become child nodes of the third child node  
    dgnNode.Children.Item(1).TransferChildren _  
        pReceivingNode:=dgnNode.Children.Item(3)  
  
End Sub
```



# Trendlines Method

Returns an object that represents a single trendline (a [Trendline](#) object) or a collection of all the trendlines (a [Trendlines](#) collection) for the series.

*object.Trendlines(Index)*

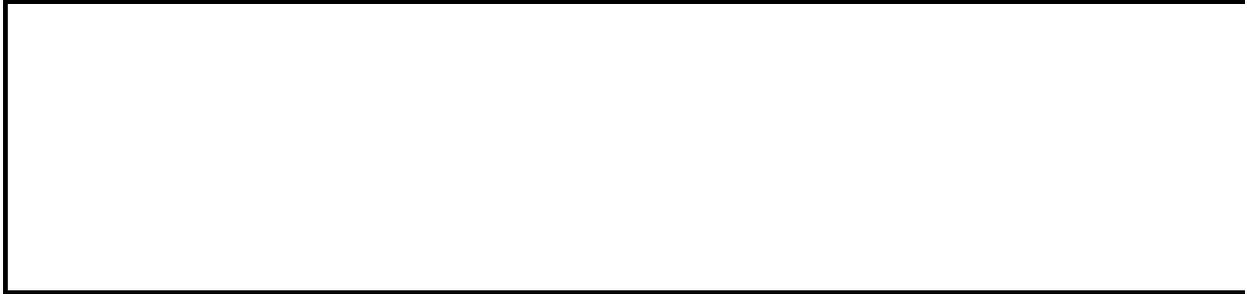
*object* Required. The **Series** object.

*Index* Optional **Variant**. The name or number of the trendline.

## Example

This example adds a linear trendline to series one in Chart1.

```
Charts("Chart1").SeriesCollection(1).Trendlines.Add Type:=xlLinear
```



[Show All](#)

# TwoColorGradient Method

 [TwoColorGradient method as it applies to the \*\*FillFormat\*\* object.](#)

Sets the specified fill to a two-color gradient.

*expression*.**TwoColorGradient**(*Style*, *Variant*)

*expression* Required. An expression that returns one of the above objects.

*Style* Required [MsoGradientStyle](#).

MsoGradientStyle can be one of these MsoGradientStyle constants.

**msoGradientDiagonalDown**

**msoGradientDiagonalUp**

**msoGradientFromCenter**

**msoGradientFromCorner**

**msoGradientFromTitle**

**msoGradientHorizontal**

**msoGradientMixed**

**msoGradientVertical**

*Variant* Required **Integer**. The gradient variant. Can be a value from 1 through 4, corresponding to one of the four variants on the **Gradient** tab in the **Fill Effects** dialog box. If *Style* is **msoGradientFromCenter**, the *Variant* argument can only be 1 or 2.

 [TwoColorGradient method as it applies to the \*\*ChartFillFormat\*\* object.](#)

Sets the specified fill to a two-color gradient.

*expression*.**TwoColorGradient**(*Style*, *Variant*)

*expression* Required. An expression that returns one of the above objects.

**Style** Required [MsoGradientStyle](#).

MsoGradientStyle can be one of these MsoGradientStyle constants.

**msoGradientDiagonalDown**

**msoGradientDiagonalUp**

**msoGradientFromCenter**

**msoGradientFromCorner**

**msoGradientFromTitle**

**msoGradientHorizontal**

**msoGradientMixed**

**msoGradientVertical**

**Variant** Required **Long**. The gradient variant. Can be a value from 1 through 4, corresponding to one of the four variants on the **Gradient** tab in the **Fill Effects** dialog box. If **Style** is **msoGradientFromCenter**, the **Variant** argument can only be 1 or 2.

## Example

This example sets the foreground color, background color, and gradient for the chart area fill on chart one.

```
With Charts(1).ChartArea.Fill
    .Visible = True
    .ForeColor.SchemeColor = 15
    .BackColor.SchemeColor = 17
    .TwoColorGradient msoGradientHorizontal, 1
End With
```



# Undo Method

Cancels the last user-interface action.

*expression*.**Undo**

*expression* Required. An expression that returns an **Application** object.

## **Remarks**

This method undoes only the last action taken by the user before running the macro, and it must be the first line in the macro. It cannot be used to undo Visual Basic commands.

## Example

This example cancels the last user-interface action. The example must be the first line in a macro.

`Application.Undo`



[Show All](#)

# Ungroup Method

 [Ungroup method as it applies to the \*\*Range\*\* object.](#)

Promotes a range in an outline (that is, decreases its outline level). The specified range must be a row or column, or a range of rows or columns. If the range is in a PivotTable report, this method ungroups the items contained in the range.

*expression*.**Ungroup**

*expression* Required. An expression that returns a **Range** object.

## Remarks

If the active cell is in a field header of a parent field, all the groups in that field are ungrouped and the field is removed from the PivotTable report. When the last group in a parent field is ungrouped, the entire field is removed from the report.

 [Ungroup method as it applies to the \*\*Shape\*\* and \*\*ShapeRange\*\* objects.](#)

Ungroups any grouped shapes in the specified shape or range of shapes. Disassembles pictures and OLE objects within the specified shape or range of shapes. Returns the ungrouped shapes as a single [ShapeRange](#) object.

*expression*.**Ungroup**

*expression* Required. An expression that returns one of the above objects.

## Remarks

Because a group of shapes is treated as a single object, grouping and ungrouping shapes changes the number of items in the **Shapes** collection and changes the index numbers of items that come after the affected items in the collection.

## Example

 [As it applies to the \*\*Range\*\* object.](#)

This example ungroups the ORDER\_DATE field.

```
Set pvtTable = Worksheets("Sheet1").Range("A3").PivotTable
Set groupRange = pvtTable.PivotFields("ORDER_DATE").DataRange
groupRange.Cells(1).Ungroup
```

 [As it applies to the \*\*Shape\*\* and \*\*ShapeRange\*\* objects.](#)

This example ungroups any grouped shapes and disassembles any pictures or OLE objects on myDocument.

```
Set myDocument = Worksheets(1)
For Each s In myDocument.Shapes
    s.Ungroup
Next
```

This example ungroups any grouped shapes on myDocument without disassembling pictures or OLE objects on the document.

```
Set myDocument = Worksheets(1)
For Each s In myDocument.Shapes
    If s.Type = msoGroup Then s.Ungroup
```



# Union Method

Returns the union of two or more ranges.

*expression*.**Union**(*Arg1*, *Arg2*, ...)

*expression* Optional. An expression that returns an **Application** object.

*Arg1*, *Arg2*, ... Required **Range**. At least two **Range** objects must be specified.

## Example

This example fills the union of two named ranges, Range1 and Range2, with the formula =RAND().

```
Worksheets("Sheet1").Activate  
Set bigRange = Application.Union(Range("Range1"), Range("Range2"))  
bigRange.Formula = "=RAND()"
```



# Unlink Method

Removes the link to a Microsoft Windows SharePoint Services site from a list.  
Returns **Nothing**.

*expression*.**Unlink()**

*expression* Required. An expression that returns a [ListObject](#) object.

## **Remarks**

After this method is called and the list is unlinked, it cannot be reversed.

## Example

The following example unlinks a list from a SharePoint site.

```
Sub UnlinkList()  
    Dim wrksht As Worksheet  
    Dim objListObj As ListObject  
  
    Set wrksht = ActiveWorkbook.Worksheets("Sheet1")  
    Set objListObj = wrksht.ListObjects(1)  
  
    objListObj.Unlink  
End Sub
```



# Unlist Method

Removes the list functionality from a [ListObject](#) object. After you use this method, the range of cells that made up the the list will be a regular range of data. Returns **Nothing**.

*expression*.**Unlist()**

*expression* Required. An expression that returns a **ListObject** object.

## Remarks

Running this method leaves the cell data, formatting, and formulas in the worksheet. The Total row is also left intact. This method removes any link to a Microsoft Windows SharePoint Services site. AutoFilter and the Insert row are also removed from the list.

## Example

The following example removes the list features from a list on a worksheet.

```
Sub DeList()  
    Dim wrksht As Worksheet  
    Dim objListObj As ListObject  
  
    Set wrksht = ActiveWorkbook.Worksheets("Sheet1")  
    Set objListObj = wrksht.ListObjects(1)  
    objListObj.Unlist  
End Sub
```



# UnMerge Method

Separates a merged area into individual cells.

*expression*.**UnMerge**

*expression* Required. An expression that returns a **Range** object.

## Example

This example separates the merged range that contains cell A3.

```
With Range("a3")
  If .MergeCells Then
    .MergeArea.UnMerge
  Else
    MsgBox "not merged"
  End If
End With
```



# Unprotect Method

Removes protection from a sheet or workbook. This method has no effect if the sheet or workbook isn't protected.

*expression*.**Unprotect**(*Password*)

*expression* Required. An expression that returns a **Chart**, **Workbook**, or **Worksheet** object.

**Password** Optional **Variant**. A string that denotes the case-sensitive password to use to unprotect the sheet or workbook. If the sheet or workbook isn't protected with a password, this argument is ignored. If you omit this argument for a sheet that's protected with a password, you'll be prompted for the password. If you omit this argument for a workbook that's protected with a password, the method fails.

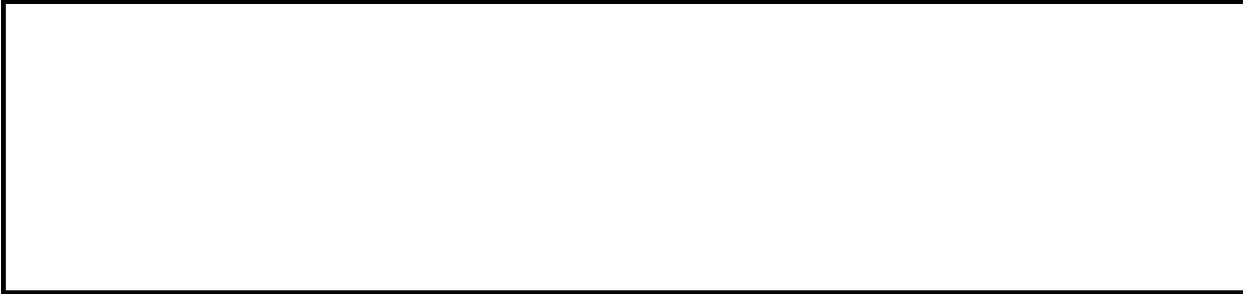
## **Remarks**

If you forget the password, you cannot unprotect the sheet or workbook. It's a good idea to keep a list of your passwords and their corresponding document names in a safe place.

## Example

This example removes protection from the active workbook.

```
ActiveWorkbook.Unprotect
```



# UnprotectSharing Method

Turns off protection for sharing and saves the workbook.

*expression*.**UnprotectSharing**(*SharingPassword*)

*expression* Required. An expression that returns a **Workbook** object.

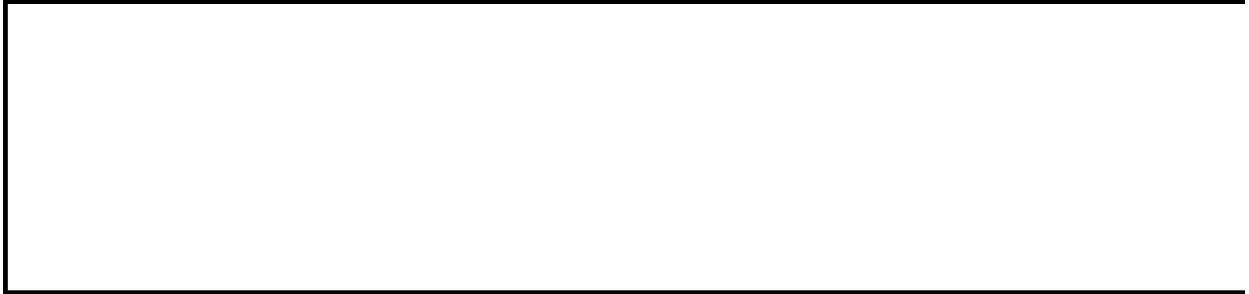
**SharingPassword** Optional **Variant**. The workbook password.

**Note** Use strong passwords that combine upper- and lowercase letters, numbers, and symbols. Weak passwords don't mix these elements. Strong password: Y6dh!et5. Weak password: House27. Use a strong password that you can remember so that you don't have to write it down.

## Example

This example turns off protection for sharing and saves the active workbook.

```
ActiveWorkbook.UnprotectSharing
```



# Update Method

Updates the link or PivotTable report.

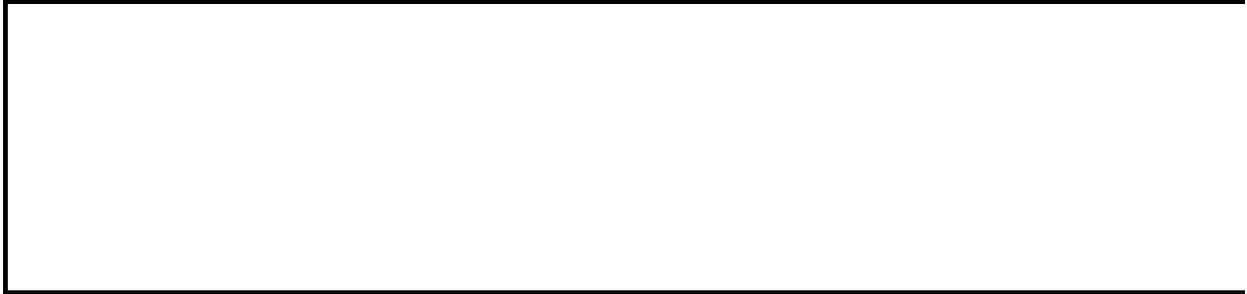
*expression*.**Update**

*expression* Required. An expression that returns an **OLEObject** or **PivotTable** object.

## Example

This example updates the link to OLE object one on Sheet1.

```
worksheets("Sheet1").OLEObjects(1).Update
```



[Show All](#)

# UpdateChanges Method

Updates the list on a Microsoft Windows SharePoint Services site with the changes made to the list in the worksheet. Returns **Nothing**.

*expression*.**UpdateChanges**(*iConflictType*)

*expression* Required. An expression that returns a **ListObject** object.

*iConflictType* Optional [XListConflict](#). Conflict resolution options.

XListConflict can be one of these XListConflict constants.

**xListConflictDialog** *default*

**xListConflictRetryAllConflicts**

**xListConflictDiscardAllConflicts**

**xListConflictError**

## **Remarks**

This method applies only to lists linked to a SharePoint site. If the SharePoint site is not available, an error is generated.

## Example

The following sample updates the list on the SharePoint site with any changes made to the default list in Sheet1 of the active workbook. The sample requires a list in the active workbook that is connected to a SharePoint site.

```
Sub UpdateListChanges()  
    Dim wrksht As Worksheet  
    Dim objListObj As ListObject  
  
    Set wrksht = ActiveWorkbook.Worksheets("Sheet1")  
    Set objListObj = wrksht.ListObjects(1)  
  
    objListObj.UpdateChanges xlListConflictDialog  
End Sub
```



# UpdateFromFile Method

Updates a read-only workbook from the saved disk version of the workbook if the disk version is more recent than the copy of the workbook that is loaded in memory. If the disk copy hasn't changed since the workbook was loaded, the in-memory copy of the workbook isn't reloaded.

*expression*.**UpdateFromFile**

*expression* Required. An expression that returns a **Workbook** object.

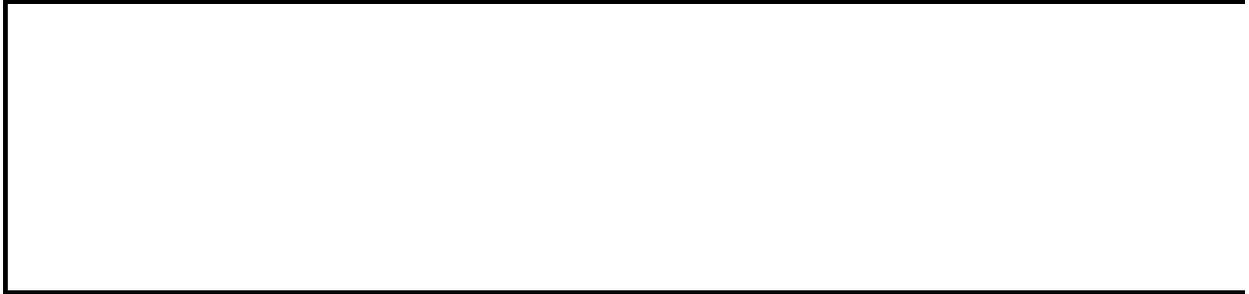
## Remarks

This method is useful when a workbook is opened as read-only by user A and opened as read/write by user B. If user B saves a newer version of the workbook to disk while user A still has the workbook open, user A cannot get the updated copy without closing and reopening the workbook and losing view settings. The **UpdateFromFile** method updates the in-memory copy of the workbook from the disk file.

## Example

This example updates the active workbook from the disk version of the file.

`ActiveWorkbook.UpdateFromFile`



[Show All](#)

# UpdateLink Method

Updates a Microsoft Excel, DDE, or OLE link (or links).

*expression*.**UpdateLink**(*Name*, *Type*)

*expression* Required. An expression that returns a **Workbook** object.

**Name** Optional **String**. The name of the Microsoft Excel or DDE/OLE link to be updated, as returned from the [LinkSources](#) method.

**Type** Optional [XILinkType](#).

XIReferenceStyle can be one of these XIReferenceStyle constants.

**xILinkTypeExcelLinks** *default*.

**xILinkTypeOLELinks** (also used for DDE links)

## Remark

**Note** When the **UpdateLink** method is called without any parameters, Excel defaults to updating all worksheet links.

## Example

This example updates all links in the active workbook.

```
ActiveWorkbook.UpdateLink Name:=ActiveWorkbook.LinkSources
```



# UpdateNotify Method

The real-time data (RTD) server uses this method to notify Microsoft Excel that new data has been received.

*expression*.**UpdateNotify()**

*expression* Required. An expression that returns an [IRTDUpdateEvent](#) object.



# UseDefaultFolderSuffix Method

Sets the folder suffix for the specified document to the default suffix for the language support you have selected or installed.

*expression*.**UseDefaultFolderSuffix**

*expression* An expression that returns a **WebOptions** object.

## Remarks

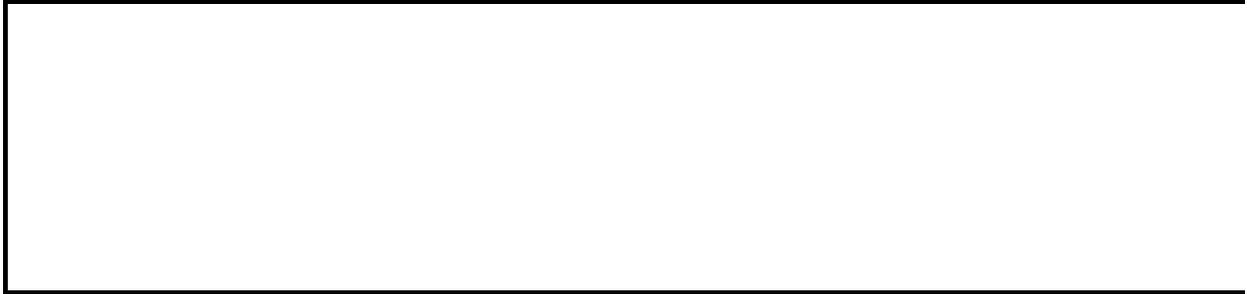
Microsoft Excel uses the folder suffix when you save a document as a Web page, use long file names, and choose to save supporting files in a separate folder (that is, if the [UseLongFileNames](#) and [OrganizeInFolder](#) properties are set to **True**).

The suffix appears in the folder name after the document name. For example, if the document is called "Book1" and the language is English, the folder name is Book1\_files. The available folder suffixes are listed in the [FolderSuffix](#) property topic.

## Example

This example sets the folder suffix for the first workbook to the default suffix.

```
Workbooks(1).WebOptions.UseDefaultFolderSuffix
```



[Show All](#)

# UserPicture Method

 [UserPicture method as it applies to the \*\*FillFormat\*\* object.](#)

Fills the specified shape with an image.

*expression*.**UserPicture**(*PictureFile*)

*expression* Required. An expression that returns one of the above objects.

**PictureFile** Required **String**. The name of the picture file.

 [UserPicture method as it applies to the \*\*ChartFillFormat\*\* object.](#)

Fills the specified shape with an image.

*expression*.**UserPicture**(*PictureFile*, *PictureFormat*, *PictureStackUnit*, *PicturePlacement*)

*expression* Required. An expression that returns one of the above objects.

**PictureFile** Optional **Variant**.

**PictureFormat** Required [XlChartPictureType](#).

XlChartPictureType can be one of these XlChartPictureType constants.

**xlStack**

**xlStackScale**

**xlStretch**

**PictureStackUnit** Required **Long**. The picture stack or scale unit (depends on the **PictureFormat** argument).

**PicturePlacement** Required [XlChartPicturePlacement](#).

XlChartPicturePlacement can be one of these XlChartPicturePlacement constants.

**xlAllFaces**

**xlEnd**

**xlEndSides**

**xlFront**

**xlFrontEnd**

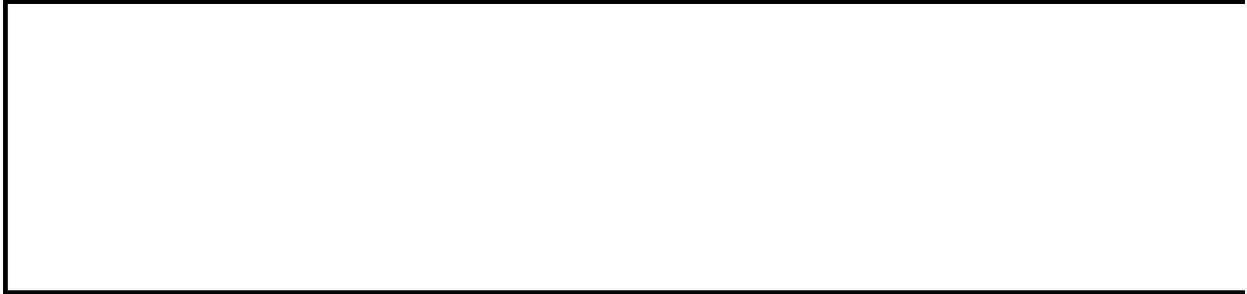
**xlFrontSides**

**xlSides**

## Example

This example sets the fill format for chart two.

```
Charts(2).ChartArea.Fill.UserPicture "brick.gif"
```



# UserTextured Method

Fills the specified shape with small tiles of an image. If you want to fill the shape with one large image, use the **UserPicture** method.

*expression*.**UserTextured**(*TextureFile*)

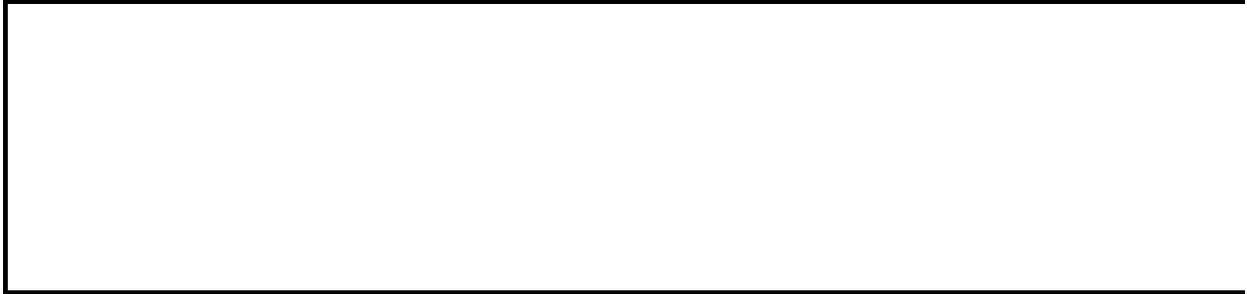
*expression* Required. An expression that returns a **FillFormat** object.

*TextureFile* Required **String**. The name of the picture file.

## Example

This example sets the fill format for chart two.

```
Charts(2).ChartArea.Fill.UserTextured "brick.gif"
```



# Verb Method

Sends a verb to the server of the specified OLE object.

*expression*.**Verb**(*Verb*)

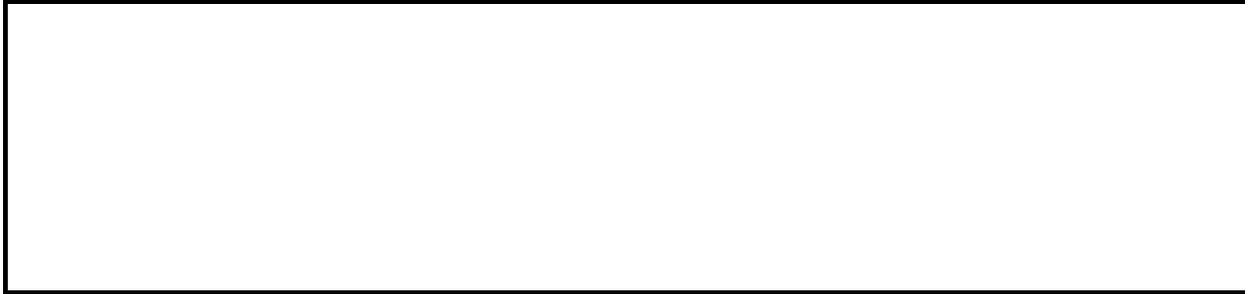
*expression* Required. An expression that returns an **OLEObject** object.

**Verb** Optional **Variant**. The verb that the server of the OLE object should act on. If this argument is omitted, the default verb is sent. The available verbs are determined by the object's source application. Typical verbs for an OLE object are Open and Primary (represented by the **XIOLEVerb** constants **xlOpen** and **xlPrimary**).

## Example

This example sends the default verb to the server for OLE object one on Sheet1.

```
worksheets("Sheet1").OLEObjects(1).Verb
```



# Volatile Method

Marks a user-defined function as volatile. A volatile function must be recalculated whenever calculation occurs in any cells on the worksheet. A nonvolatile function is recalculated only when the input variables change. This method has no effect if it's not inside a user-defined function used to calculate a worksheet cell.

*expression*.**Volatile**(*Volatile*)

*expression* Required. An expression that returns an **Application** object.

**Volatile** Optional **Variant**. **True** to mark the function as volatile. **False** to mark the function as nonvolatile. The default value is **True**

## Example

This example marks the user-defined function "My\_Func" as volatile. The function will be recalculated whenever calculation occurs in any cells on the worksheet on which this function appears.

```
Function My_Func()  
    Application.Volatile  
    '  
    '    Remainder of the function  
    ">  
End Function
```



# Wait Method

Pauses a running macro until a specified time. Returns **True** if the specified time has arrived.

**Important** The **Wait** method suspends all Microsoft Excel activity and may prevent you from performing other operations on your computer while **Wait** is in effect. However, background processes such as printing and recalculation continue.

*expression*.**Wait**(*Time*)

*expression* Required. An expression that returns an **Application** object.

**Time** Required **Variant**. The time at which you want the macro to resume, in Microsoft Excel date format.

## Example

This example pauses a running macro until 6:23 P.M. today.

```
Application.Wait "18:23:00"
```

This example pauses a running macro for approximately 10 seconds.

```
newHour = Hour(Now())  
newMinute = Minute(Now())  
newSecond = Second(Now()) + 10  
waitTime = TimeSerial(newHour, newMinute, newSecond)  
Application.Wait waitTime
```

This example displays a message indicating whether 10 seconds have passed.

```
If Application.Wait(Now + TimeValue("0:00:10")) Then  
    MsgBox "Time expired"  
End If
```



# WebPagePreview Method

Displays a preview of the specified workbook as it would look if saved as a Web page.

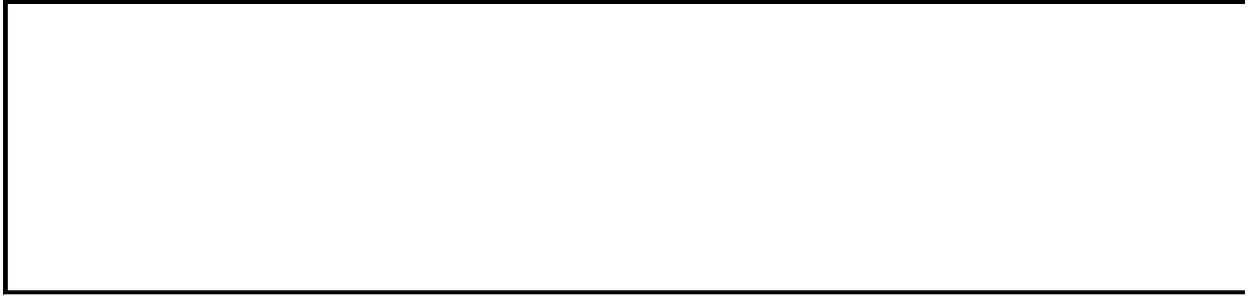
*expression*.**WebPagePreview**

*expression* An expression that returns a **Workbook** object.

## Example

This example displays a preview of the first workbook as a Web page.

workbooks(1).WebPagePreview



# XmlDataQuery Method

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Returns a [Range](#) object that represents the cells mapped to a particular XPath. Returns **Nothing** if the specified XPath has not been mapped to the worksheet, or if the mapped range is empty.

*expression.XmlDataQuery(XPath, SelectionNamespaces, Map)*

*expression* Required. An expression that returns a [Worksheet](#) object.

*XPath* Required **String**. The XPath to query for.

*SelectionNamespaces* Optional **String**. A space-delimited **String** that contains the namespaces referenced in the XPath parameter. A run-time error will be generated if one of the specified namespaces cannot be resolved.

*Map* Optional [XmlMap](#). Specify an XML map if you want to query for the XPath within a specific map.

## Remarks

If the XPath exists within a column in an XML list, the **Range** object returned does not include the header row or the Insert row.

--

[Show All](#)

# XmlImport Method

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Imports an XML data file into the current workbook. Returns [XlXmlImportResult](#).

XlXmlImportResult can be one of the following XlXmlImportResult constants

**xlXmlImportElementsTruncated** The contents of the specified XML data file have been truncated because the XML data file is too large for the worksheet.

**xlXmlImportSuccess** The XML data file was successfully imported.

**xlXmlImportValidationFailed** The contents of the XML data file do not match the specified schema map. The file was not imported.

*expression*.**XmlImport**(*Url*, *ImportMap*, *Overwrite*, *Destination*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*Url* Required **String**. A uniform resource locator (URL) or a uniform naming convention (UNC) path to a XML data file.

*ImportMap* Required [XmlMap](#). The schema map to apply when importing the file.

*Overwrite* Optional **Boolean**. If a value is not specified for the **Destination** parameter, then this parameter specifies whether or not to overwrite data that has been mapped to the schema map specified in the **ImportMap** parameter. Set to **True** to overwrite the data or **False** to append the new data to the existing data. The default value is **True**. If a value is specified for the **Destination** parameter, then this parameter specifies whether or not to overwrite existing data. Set to **True** to overwrite existing data or **False** to cancel the import if data would be overwritten. The default value is **True**.

***Destination*** Optional **Range**. The data will be imported into a new XML list at the range specified.

## Remarks

Don't specify a value for the *Destination* parameter if you want to import data into an existing mapping.

The following conditions will cause the **XMLImport** method to generate run-time errors:

- The specified XML data contains syntax errors.
- The import process was cancelled because the specified data cannot fit into the worksheet.

Use the [XmlImportXml](#) method of the [Workbook](#) object to import XML data that has been previously loaded into memory.

---

[Show All](#)

# XmlImportXml Method

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Imports an XML data stream that has been previously loaded into memory. Returns [XlXmlImportResult](#).

XlXmlImportResult can be one of the following XlXmlImportResult constants

**xlXmlImportElementsTruncated** The contents of the specified XML data file have been truncated because the XML data file is too large for the worksheet.

**xlXmlImportSuccess** The XML data file was successfully imported.

**xlXmlImportValidationFailed** The contents of the XML data file do not match the specified schema map. The file was not imported.

*expression.XmlImportXml(Data, ImportMap, Overwrite, Destination)*

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Data** Required **String**. The data to import.

**ImportMap** Required [XmlMap](#). The schema map to apply when importing the file.

**Overwrite** Optional **Boolean**. If a value is not specified for the **Destination** parameter, then this parameter specifies whether or not to overwrite data that has been mapped to the schema map specified in the **ImportMap** parameter. Set to **True** to overwrite the data or **False** to append the new data to the existing data. The default value is **True**. If a value is specified for the **Destination** parameter, then this parameter specifies whether or not to overwrite existing data. Set to **True** to overwrite existing data or **False** to cancel the import if data would be overwritten. The default value is **True**.

**Destination** Optional [Range](#). The data will be imported into a new XML list in

the range specified.

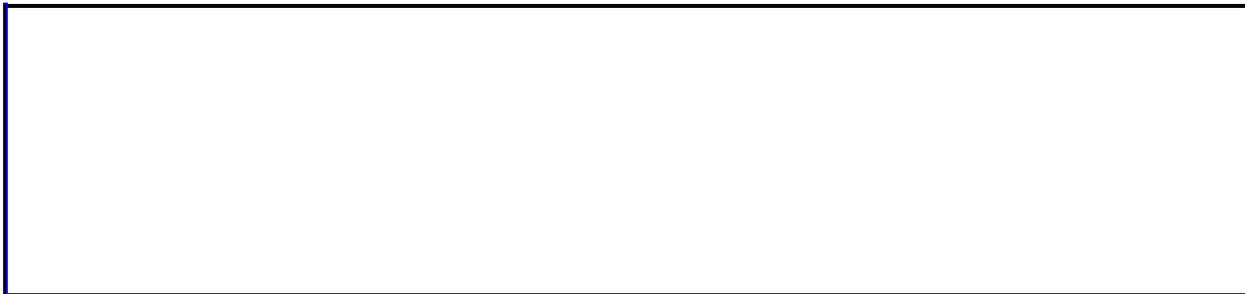
## Remarks

Don't specify a value for the *Destination* parameter if you want to import data into an existing mapping.

The following conditions will cause the **XMLImport** method to generate run-time errors:

- The specified XML data contains syntax errors.
- The import process was cancelled because the specified data cannot fit into the worksheet.

Use the [XmlImport](#) method of the [Workbook](#) object to import an XML data file into the current workbook.



[Show All](#)

# XmlMapQuery Method

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Returns a [Range](#) object that represents the cells mapped to a particular XPath. Returns **Nothing** if the specified XPath has not been mapped to the worksheet.

*expression.XmlMapQuery(XPath, SelectionNamespaces, Map)*

*expression* Required. An expression that returns a **Worksheet** object.

*XPath* Required **String**. The XPath to query for.

*SelectionNamespaces* Optional **String**. A space-delimited **String** that contains the namespaces referenced in the XPath parameter. A run-time error will be generated if one of the specified namespaces cannot be resolved.

*Map* Optional [XmlMap](#). Specify an XML map if you want to query for the XPath within a specific map.

## Remarks

Unlike the [XmlDataQuery](#) method, the **XmlMapQuery** method returns the entire column of an XML list, including the header row and the [Insert row](#).

---

# XYGroups Method

On a 2-D chart, returns an object that represents either a single scatter chart group (a [ChartGroup](#) object) or a collection of the scatter chart groups (a [ChartGroups](#) collection).

*expression*.XYGroups(*Index*)

*expression* Required. An expression that returns a **Chart** object.

*Index* Optional **Variant**. Specifies the chart group.

## Example

This example sets X-Y group (scatter group) one to use a different color for each data marker. The example should be run on a 2-D chart.

```
Charts("Chart1").XYGroups(1).VaryByCategories = True
```



[Show All](#)

# ZOrder Method

Moves the specified shape in front of or behind other shapes in the collection (that is, changes the shape's position in the z-order).

*expression.ZOrder(ZOrderCmd)*

*expression* Required. An expression that returns one of the objects in the Applies To list.

**ZOrderCmd** Required [MsoZOrderCmd](#). Specifies where to move the specified shape relative to the other shapes.

MsoZOrderCmd can be one of these MsoZOrderCmd constants.

**msoBringForward**

**msoBringInFrontOfText**. Used only in Microsoft Word.

**msoBringToFront**

**msoSendBackward**

**msoSendBehindText**. Used only in Microsoft Word.

**msoSendToBack**

## Remarks

Use the [ZOrderPosition](#) property to determine a shape's current position in the z-order.

## Example

This example adds an oval to myDocument and then places the oval second from the back in the z-order if there is at least one other shape on the document.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddShape(msoShapeOval, 100, 100, 100, 300)
    While .ZOrderPosition > 2
        .ZOrder msoSendBackward
    Wend
End With
```



[Show All](#)

# Accent Property

Allows the user to place a vertical accent bar to separate the callout text from the callout line. Read/write [MsoTriState](#).

MsoTriState can be one of these MsoTriState constants.

**msoCTrue**

**msoFalse**

**msoTriStateMixed**

**msoTriStateToggle**

**msoTrue** A vertical accent bar separates the callout text from the callout line.

*expression*.**Accent**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example adds to myDocument an oval and a callout that points to the oval. The callout text won't have a border, but it will have a vertical accent bar that separates the text from the callout line.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes
    .AddShape msoShapeOval, 180, 200, 280, 130
    With .AddCallout(msoCalloutTwo, 420, 170, 170, 40)
        .TextFrame.Characters.Text = "My oval"
        With .Callout
            .Accent = msoTrue
            .Border = False
        End With
    End With
End With
```



# AcceptLabelsInFormulas Property

**True** if labels can be used in worksheet formulas. The default value is **False**.  
Read/write **Boolean**.

*expression*.**AcceptLabelsInFormula**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example sets the **AcceptLabelsInFormulas** property for the active workbook and then sets cells B1:D1 on worksheet one to be column labels.

```
ActiveWorkbook.AcceptLabelsInFormulas = True  
Worksheets(1).Range("b1:d1").FormulaLabel = xlColumnLabels
```



[Show All](#)

# Active Property

Returns a **Boolean** value indicating whether a **ListObject** object in a worksheet is active— that is, whether the active cell is inside the range of the **ListObject** object. Read-only **Boolean**.

*expression.Active*

*expression* Required. An expression that returns a **ListObject** object.

## Remarks

Because there is no **Activate** method for the **ListObject** object, you can activate a **ListObject** object only by activating a cell range within the [list](#).

## Example

The following example activates the list in the default **ListObject** object in the first worksheet of the active workbook. Invoking the **Activate** method of the **Range** object without cell parameters activates the entire range for the list.

```
Function MakeListActive() As Boolean
    Dim wrksht As Worksheet
    Dim objList As ListObject

    Set wrksht = ActiveWorkbook.Worksheets("Sheet1")
    Set objList = wrksht.ListObjects(1)
    objList.Range.Activate

    MakeListActive = objList.Active
End Function
```



# ActiveCell Property

Returns a [Range](#) object that represents the active cell in the active window (the window on top) or in the specified window. If the window isn't displaying a worksheet, this property fails. Read-only.

## Remarks

If you don't specify an object qualifier, this property returns the active cell in the active window.

Be careful to distinguish between the active cell and the selection. The active cell is a single cell inside the current selection. The selection may contain more than one cell, but only one is the active cell.

The following expressions all return the active cell, and are all equivalent.

```
ActiveCell  
Application.ActiveCell  
ActiveWindow.ActiveCell  
Application.ActiveWindow.ActiveCell
```

## Example

This example uses a message box to display the value in the active cell. Because the **ActiveCell** property fails if the active sheet isn't a worksheet, the example activates Sheet1 before using the **ActiveCell** property.

```
Worksheets("Sheet1").Activate  
MsgBox ActiveCell.Value
```

This example changes the font formatting for the active cell.

```
Worksheets("Sheet1").Activate  
With ActiveCell.Font  
    .Bold = True  
    .Italic = True  
End With
```



# ActiveChart Property

Returns a [Chart](#) object that represents the active chart (either an embedded chart or a chart sheet). An embedded chart is considered active when it's either selected or activated. When no chart is active, this property returns **Nothing**.  
Read-only.

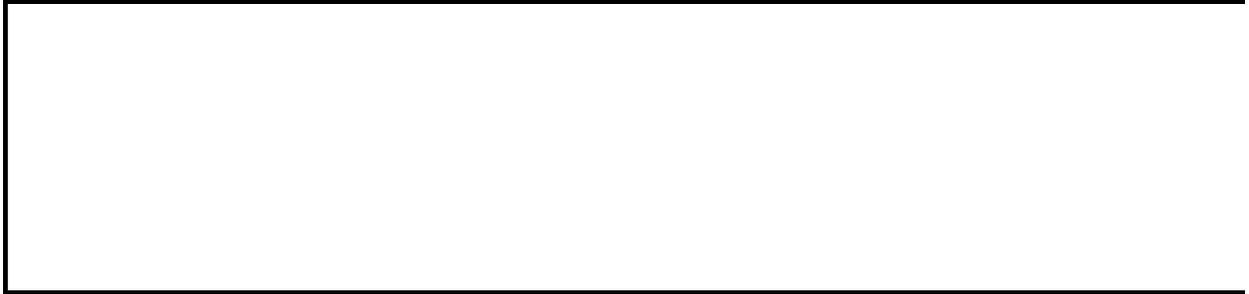
## Remarks

If you don't specify an object qualifier, this property returns the active chart in the active workbook.

## Example

This example turns on the legend for the active chart.

```
ActiveChart.HasLegend = True
```



# ActivePane Property

Returns a [Pane](#) object that represents the active pane in the window. Read-only.

## Remarks

This property can be used only on worksheets and macro sheets.

This property returns a **Pane** object. You must use the [Index](#) property to obtain the index of the active pane.

## Example

This example activates the next pane of the active window in Book1.xls. You cannot activate the next pane if the panes are frozen. The example must be run from a workbook other than Book1.xls. Before running the example, make sure that Book1.xls has either two or four panes in the active worksheet.

```
Workbooks("BOOK1.XLS").Activate
If not ActiveWindow.FreezePanes Then
    With ActiveWindow
        i = .ActivePane.Index
        If i = .Panes.Count Then
            .Panes(1).Activate
        Else
            .Panes(i+1).Activate
        End If
    End With
End If
```



# ActivePrinter Property

Returns or sets the name of the active printer. Read/write **String**.

## Example

This example displays the name of the active printer.

```
MsgBox "The name of the active printer is " & _  
Application.ActivePrinter
```



# ActiveSheet Property

Returns an object that represents the active sheet (the sheet on top) in the active workbook or in the specified window or workbook. Returns **Nothing** if no sheet is active. Read-only.

## Remarks

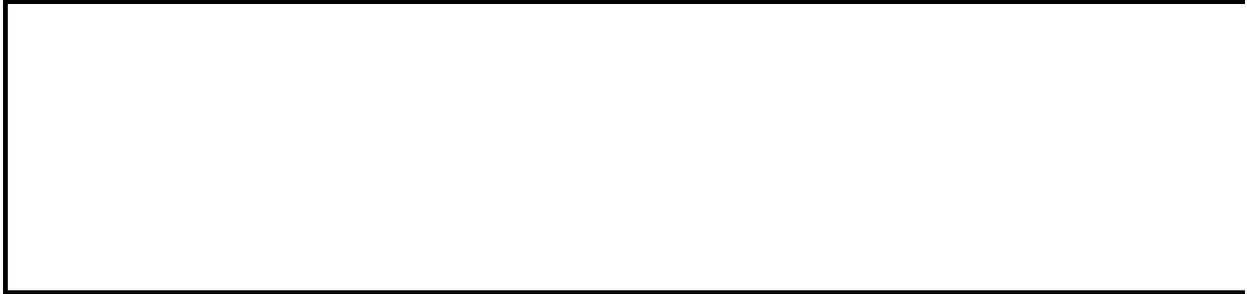
If you don't specify an object qualifier, this property returns the active sheet in the active workbook.

If a workbook appears in more than one window, the **ActiveSheet** property may be different in different windows.

## Example

This example displays the name of the active sheet.

```
MsgBox "The name of the active sheet is " & ActiveSheet.Name
```



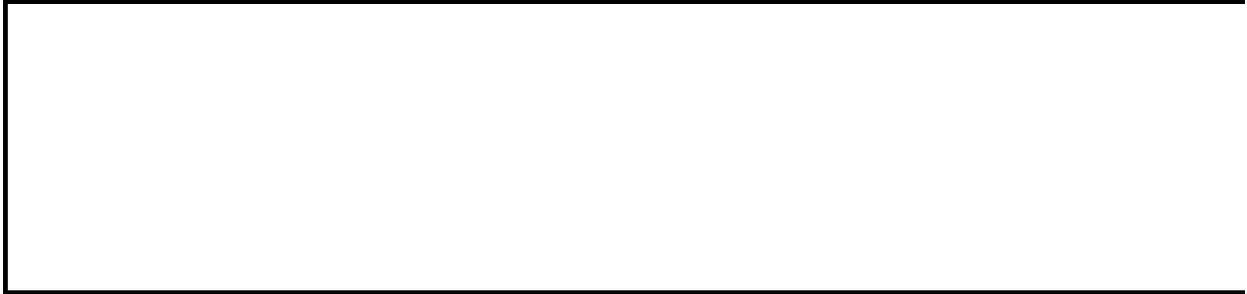
# ActiveWindow Property

Returns a [Window](#) object that represents the active window (the window on top). Read-only. Returns **Nothing** if there are no windows open.

## Example

This example displays the name (**Caption** property) of the active window.

```
MsgBox "The name of the active window is " & ActiveWindow.Caption
```



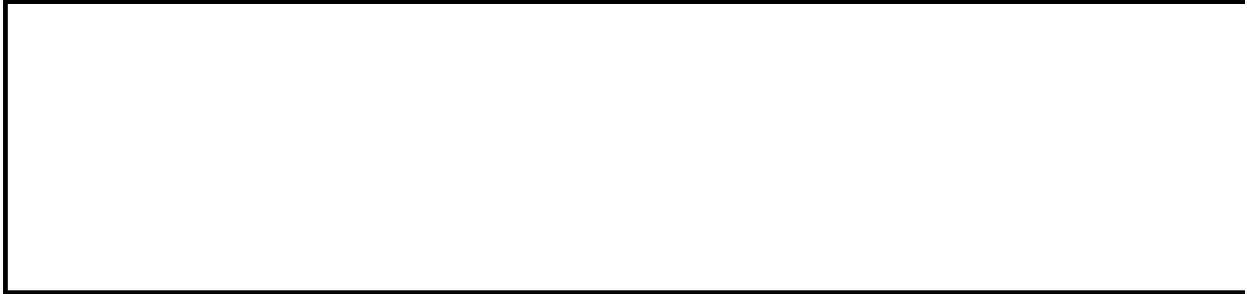
# ActiveWorkbook Property

Returns a [Workbook](#) object that represents the workbook in the active window (the window on top). Read-only. Returns **Nothing** if there are no windows open or if either the Info window or the Clipboard window is the active window.

## Example

This example displays the name of the active workbook.

```
MsgBox "The name of the active workbook is " & ActiveWorkbook.Name
```



# ActiveXControl Property

Returns an **Object** that represents an ActiveX control displayed in the **Document Actions** task pane. Read-only.

*expression*.**ActiveXControl**

*expression* Required. An expression that returns a **SmartTagAction** object.

## Remarks

For more information on smart documents, see the Smart Document Software Development Kit on the [Microsoft Developer Network \(MSDN\)](#) Web site.

---

---

---

[Show All](#)

# AddIndent Property

[AddIndent property as it applies to the \*\*Style\*\* object.](#)

**True** if text is automatically indented when the text alignment in a cell is set to equal distribution either horizontally or vertically. Read/write **Boolean**.

*expression*.**AddIndent**

*expression* Required. An expression that returns a **Style** object.

[AddIndent property as it applies to the \*\*CellFormat\*\* and \*\*Range\*\* objects.](#)

**True** if text is automatically indented when the text alignment in a cell is set to equal distribution either horizontally or vertically. Read/write **Variant**.

*expression*.**AddIndent**

*expression* Required. An expression that returns one of the above objects.

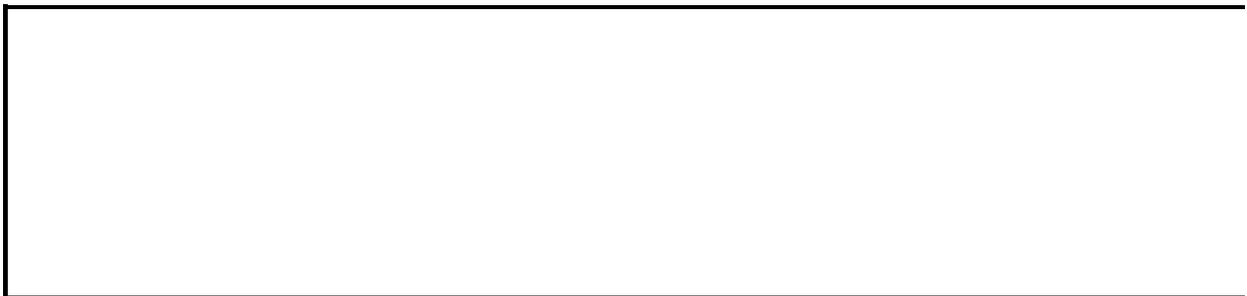
## Remarks

To set text alignment to equal distribution, you can set the [VerticalAlignment](#) property to **xlVAlignDistributed** when the value of the [Orientation](#) property is **xlVertical**, and you can set the [HorizontalAlignment](#) property to **xlHAlignDistributed** when the value of the **Orientation** property is **xlHorizontal**.

## Example

This example sets the horizontal alignment for text in cell A1 on Sheet1 to equal distribution and then indents the text.

```
With Worksheets("Sheet1").Range("A1")  
    .HorizontalAlignment = xlHAlignDistributed  
    .AddIndent = True  
End With
```



# AddIns Property

Returns an [AddIns](#) collection that represents all the add-ins listed in the **Add-Ins** dialog box (**Tools** menu). Read-only.

For information about returning a single member of a collection, see [Returning an Object from a Collection](#).

## Remarks

Using this method without an object qualifier is equivalent to `Application.Addins`.

## Example

This example displays the status of the Analysis ToolPak add-in. Note that the string used as the index to the **AddIns** collection is the title of the add-in, not the add-in's file name.

```
If AddIns("Analysis ToolPak").Installed = True Then
    MsgBox "Analysis ToolPak add-in is installed"
Else
    MsgBox "Analysis ToolPak add-in is not installed"
End If
```



[Show All](#)

# Address Property

 [Address property as it applies to the \*\*Hyperlink\*\* object.](#)

Returns or sets the address of the target document. Read/write **String**.

*expression*.**Address**

*expression* Required. An expression that returns one of the above objects.

 [Address property as it applies to the \*\*Range\*\* object.](#)

Returns the range reference in the language of the macro. Read-only **String**.

*expression*.**Address**(*RowAbsolute*, *ColumnAbsolute*, *ReferenceStyle*, *External*, *RelativeTo*)

*expression* Required. An expression that returns one of the above objects.

**RowAbsolute** Optional **Variant**. **True** to return the row part of the reference as an absolute reference. The default value is **True**.

**ColumnAbsolute** Optional **Variant**. **True** to return the column part of the reference as an absolute reference. The default value is **True**.

**ReferenceStyle** Optional [XlReferenceStyle](#).

XlReferenceStyle can be one of these XlReferenceStyle constants.

**xIA1** *default*. Use **xIA1** to return an A1-style reference.

**xIR1C1**. Use **xIR1C1** to return an R1C1-style reference.

**External** Optional **Variant**. **True** to return an external reference. **False** to return a local reference. The default value is **False**.

**RelativeTo** Optional **Variant**. If **RowAbsolute** and **ColumnAbsolute** are **False**, and **ReferenceStyle** is **xIR1C1**, you must include a starting point for the relative reference. This argument is a **Range** object that defines the starting point.

## Remarks

If the reference contains more than one cell, ***RowAbsolute*** and ***ColumnAbsolute*** apply to all rows and columns.

## Example

The following example displays four different representations of the same cell address on Sheet1. The comments in the example are the addresses that will be displayed in the message boxes.

```
Set mc = Worksheets("Sheet1").Cells(1, 1)
MsgBox mc.Address() ' $A$1
MsgBox mc.Address(RowAbsolute:=False) ' $A1
MsgBox mc.Address(ReferenceStyle:=xlR1C1) ' R1C1
MsgBox mc.Address(ReferenceStyle:=xlR1C1, _
    RowAbsolute:=False, _
    ColumnAbsolute:=False, _
    RelativeTo:=Worksheets(1).Cells(3, 3)) ' R[-2]C[-2]
```



[Show All](#)

# AddressLocal Property

Returns the range reference for the specified range in the language of the user.  
Read-only **String**.

*expression*.**AddressLocal**(*RowAbsolute*, *ColumnAbsolute*, *ReferenceStyle*,  
*External*, *RelativeTo*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**RowAbsolute** Optional **Variant**. **True** to return the row part of the reference as an absolute reference. The default value is **True**.

**ColumnAbsolute** Optional **Variant**. **True** to return the column part of the reference as an absolute reference. The default value is **True**.

**ReferenceStyle** Optional [XlReferenceStyle](#).

XlReferenceStyle can be one of these XlReferenceStyle constants.

**xIA1** *default*. Use **xIA1** to return an A1-style reference

**xIR1C1**. Use **xIR1C1** to return an R1C1-style reference.

**External** Optional **Variant**. **True** to return an external reference. **False** to return a local reference. The default value is **False**.

**RelativeTo** Optional **Variant**. If **RowAbsolute** and **ColumnAbsolute** are both set to **False** and **ReferenceStyle** is set to **xIR1C1**, you must include a starting point for the relative reference. This argument is a **Range** object that defines the starting point for the reference.

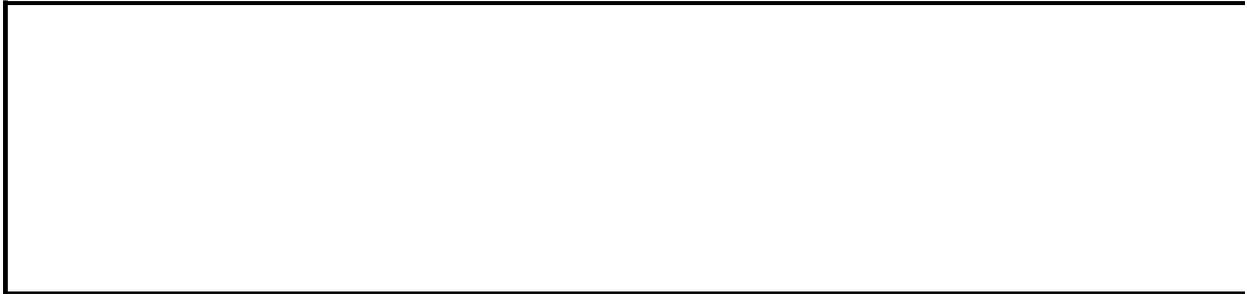
## Remarks

If the reference contains more than one cell, ***RowAbsolute*** and ***ColumnAbsolute*** apply to all rows and all columns, respectively.

## Example

Assume that this example was created using U.S. English language support and was then run in using German language support. The example displays the text shown in the comments.

```
Set mc = Worksheets(1).Cells(1, 1)
MsgBox mc.AddressLocal() ' $$$1
MsgBox mc.AddressLocal(RowAbsolute:=False) ' $A1
MsgBox mc.AddressLocal(ReferenceStyle:=xlR1C1) ' Z1S1
MsgBox mc.AddressLocal(ReferenceStyle:=xlR1C1, _
    RowAbsolute:=False, _
    ColumnAbsolute:=False, _
    RelativeTo:=Worksheets(1).Cells(3, 3)) ' Z(-2)S(-2)
```



# AdjustColumnWidth Property

**True** if the column widths are automatically adjusted for the best fit each time you refresh the specified query table or XML map. **False** if the column widths aren't automatically adjusted with each refresh. The default value is **True**.

Read/write **Boolean**.

## Remarks

The maximum column width is two-thirds the width of the screen.

## Example

This example turns off automatic column-width adjustment for the newly added query table on the first worksheet in the first workbook.

```
With Workbooks(1).Worksheets(1).QueryTables _  
    .Add(Connection:= varDBConnStr, _  
        Destination:=Range("B1"), _  
        Sql:="Select Price From CurrentStocks " & _  
            "Where Symbol = 'MSFT'")  
    .AdjustColumnWidth = False  
    .Refresh  
End With
```



# Adjustments Property

Returns an [Adjustments](#) object that contains adjustment values for all the adjustments in the specified shape. Applies to any **Shape** or **ShapeRange** object that represents an AutoShape, WordArt, or a connector. Read-only.

## Example

This example sets to 0.25 the value of adjustment one on shape one on myDocument.

```
Set myDocument = Worksheets(1)  
myDocument.Shapes(1).Adjustments(1) = 0.25
```



# ADOConnection Property

Returns an ADO connection object if the PivotTable cache is connected to an OLE DB data source. The **ADOConnection** property exposes Microsoft Excel's connection to the data provider allowing the user to write code within the context of the same session that Excel is using with ADO (relational source) or ADOMD (OLAP source). Read-only.

*expression*.**ADOConnection**

*expression* Required. An expression that returns one a [PivotCache](#) object.

## Remarks

The **ADOConnection** property is available only for sessions where the data source is an OLE DB data source. When there is no ADO session the query will result in a run-time error.

The **ADOConnection** property can be used for any OLEDB-based cache with ADO. The ADO connection object can be used with ADOMD for finding information about OLAP Cubes on which the cache is based.

## Example

This example sets an ADODB Connection object to the **ADOConnection** property of the PivotTable cache. The example assumes a PivotTable report exists on the active worksheet.

```
Sub UseADOConnection()  
  
    Dim ptOne As PivotTable  
    Dim cmdOne As New ADODB.Command  
    Dim cfOne As CubeField  
  
    Set ptOne = Sheet1.PivotTables(1)  
    ptOne.PivotCache.MaintainConnection = True  
    Set cmdOne.ActiveConnection = ptOne.PivotCache.ADOConnection  
  
    ptOne.PivotCache.MakeConnection  
  
    ' Create a set.  
    cmdOne.CommandText = "Create Set [Warehouse].[My Set] as '{{[Prod  
    cmdOne.CommandType = adCmdUnknown  
    cmdOne.Execute  
  
    ' Add a set to the CubeField.  
    Set cfOne = ptOne.CubeFields.AddSet("My Set", "My Set")  
  
End Sub
```

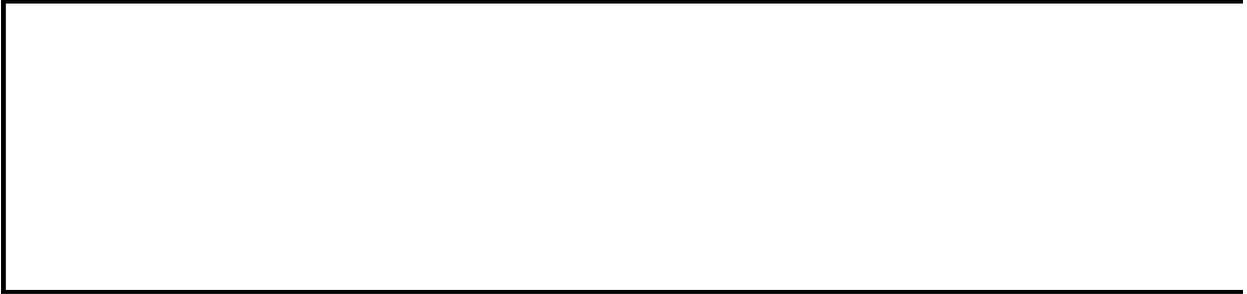
This example adds a calculated member, assuming a PivotTable report exists on the active worksheet.

```
Sub AddMember()  
  
    Dim cmd As New ADODB.Command  
  
    If Not ActiveSheet.PivotTables(1).PivotCache.IsConnected Then  
        ActiveSheet.PivotTables(1).PivotCache.MakeConnection  
    End If  
  
    Set cmd.ActiveConnection = ActiveSheet.PivotTables(1).PivotCache  
  
    ' Add a calculated member.  
    cmd.CommandText = "CREATE MEMBER [Warehouse].[Product].[All Prod  
    cmd.CommandType = adCmdUnknown
```

```
cmd.Execute
```

```
ActiveSheet.PivotTables(1).PivotCache.Refresh
```

```
End Sub
```



# AlertBeforeOverwriting Property

**True** if Microsoft Excel displays a message before overwriting nonblank cells during a drag-and-drop editing operation. Read/write **Boolean**.

## Example

This example causes Microsoft Excel to display an alert before overwriting nonblank cells during drag-and-drop editing.

```
Application.AlertBeforeOverwriting = True
```



[Show All](#)

# AlertStyle Property

Returns the validation alert style. Read-only [XIDVAlertStyle](#).

XIDVAlertStyle can be one of these XIDVAlertStyle constants.

**xlValidAlertInformation**

**xlValidAlertStop**

**xlValidAlertWarning**

*expression.AlertStyle*

*expression* Required. An expression that returns one of the objects in the Applies To list.

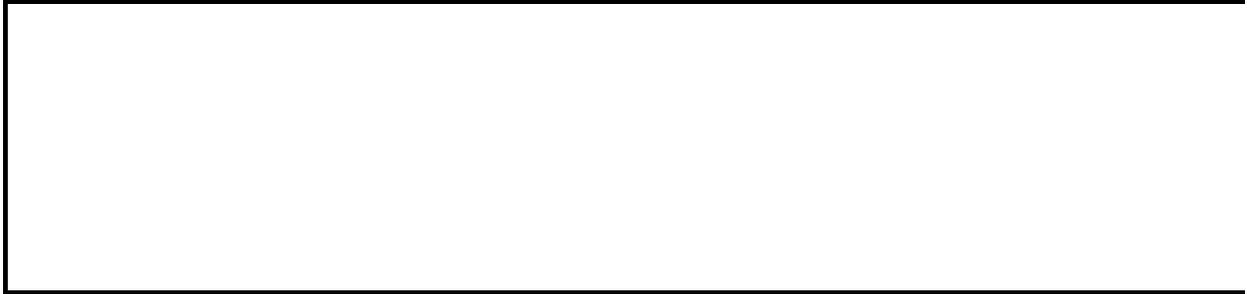
## Remarks

Use the **Add** method to set the alert style for a range. If the range already has data validation, use the **Modify** method to change the alert style.

## Example

This example displays the alert style for cell E5.

```
MsgBox Range("e5").Validation.AlertStyle
```



[Show All](#)

# Alignment Property

 [Alignment property as it applies to the \*\*TextEffectFormat\*\* object.](#)

Returns or sets the alignment for WordArt. Read/write **MsoTextEffectAlignment**.

MsoTextEffectAlignment can be one of these MsoTextEffectAlignment constants.

**msoTextEffectAlignmentCentered**

**msoTextEffectAlignmentLeft**

**msoTextEffectAlignmentLetterJustify**

**msoTextEffectAlignmentMixed**

**msoTextEffectAlignmentRight**

**msoTextEffectAlignmentStretchJustify**

**msoTextEffectAlignmentWordJustify**

*expression*.**Alignment**

*expression* Required. An expression that returns one of the above objects.

 [Alignment property as it applies to the \*\*Phonetic\*\*, \*\*Phonetics\*\*, and \*\*TickLabels\*\* objects.](#)

Returns or sets the alignment for the specified phonetic text or tick label. Read/write **Long**.

*expression*.**Alignment**

*expression* Required. An expression that returns one of the above objects.

Phoentic or Phonetics can be one of these XlPhoneticAlignment constants.

**XlPhoneticAlignCenter**

**XlPhoneticAlignDistributed**

**XlPhoneticAlignLeft**

**XIPhoneticAlignNoControl**

TickLabels can be one of these XIHAlign constants.

**XIHAlignCenter**

**XIHAlignLeft**

**XIHAlignRight**

## Example

This example adds a WordArt object to worksheet one and then right aligns the WordArt.

```
Set mySh = Worksheets(1).Shapes
Set myTE = mySh.AddTextEffect(PresetTextEffect:=msoTextEffect1, _
    Text:="Test Text", FontName:="Palatino", FontSize:=54, _
    FontBold:=True, FontItalic:=False, Left:=100, Top:=50)
myTE.TextEffect.Alignment = msoTextEffectAlignmentRight
```



# AllowDeletingColumns Property

Returns **True** if the deletion of columns is allowed on a protected worksheet.  
Read-only **Boolean**.

*expression*.**AllowDeletingColumns**

*expression* Required. An expression that returns a [Protection](#) object.

## Remarks

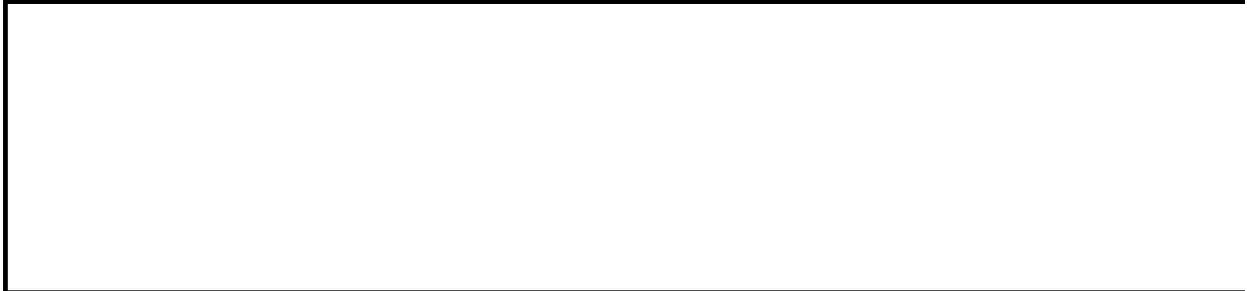
The **AllowDeletingColumns** property can be set by using the [Protect](#) method arguments.

The columns containing the cells to be deleted must be unlocked when the sheet is protected.

## Example

This example unlocks column A then allows the user to delete column A on the protected worksheet and notifies the user.

```
Sub ProtectionOptions()  
    ActiveSheet.Unprotect  
  
    'Unlock column A.  
    Columns("A:A").Locked = False  
  
    ' Allow column A to be deleted on a protected worksheet.  
    If ActiveSheet.Protection.AllowDeletingColumns = False Then  
        ActiveSheet.Protect AllowDeletingColumns:=True  
    End If  
  
    MsgBox "Column A can be deleted on this protected worksheet."  
  
End Sub
```



# AllowDeletingRows Property

Returns **True** if the deletion of rows is allowed on a protected worksheet. Read-only **Boolean**.

*expression*.**AllowDeletingRows**

*expression* Required. An expression that returns a [Protection](#) object.

## Remarks

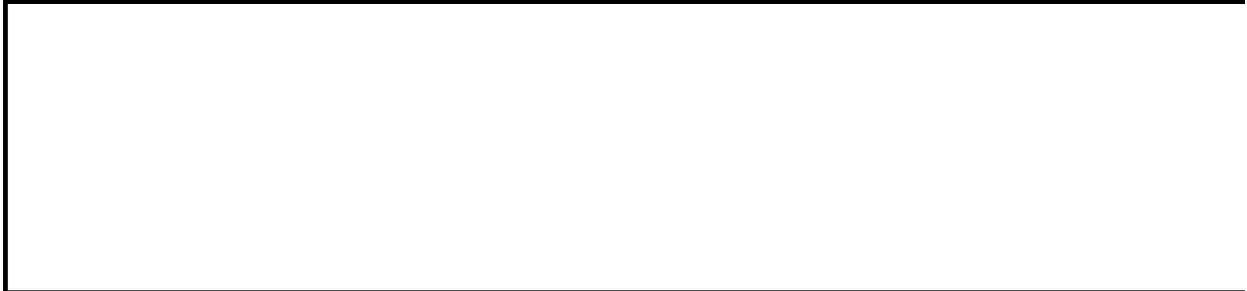
The **AllowDeletingRows** property can be set by using the [Protect](#) method arguments.

The rows containing the cells to be deleted must be unlocked when the sheet is protected.

## Example

This example unlocks row 1 then allows the user to delete row 1 on the protected worksheet and notifies the user.

```
Sub ProtectionOptions()  
    ActiveSheet.Unprotect  
  
    'Unlock row 1.  
    Rows("1:1").Locked = False  
  
    ' Allow row 1 to be deleted on a protected worksheet.  
    If ActiveSheet.Protection.AllowDeletingRows = False Then  
        ActiveSheet.Protect AllowDeletingRows:=True  
    End If  
  
    MsgBox "Row 1 can be deleted on this protected worksheet."  
  
End Sub
```



[Show All](#)

# AllowEdit Property

[AllowEdit property as it applies to the \*\*UserAccess\*\* object.](#)

**True** if the user is allowed access to the specified range on a protected worksheet. Read/write **Boolean**.

*expression*.**AllowEdit**

*expression* Required. An expression that returns a **UserAccess** object.

[AllowEdit property as it applies to the \*\*Range\*\* object.](#)

**True** if the range can be edited on a protected worksheet. Read-only **Boolean**.

*expression*.**AllowEdit**

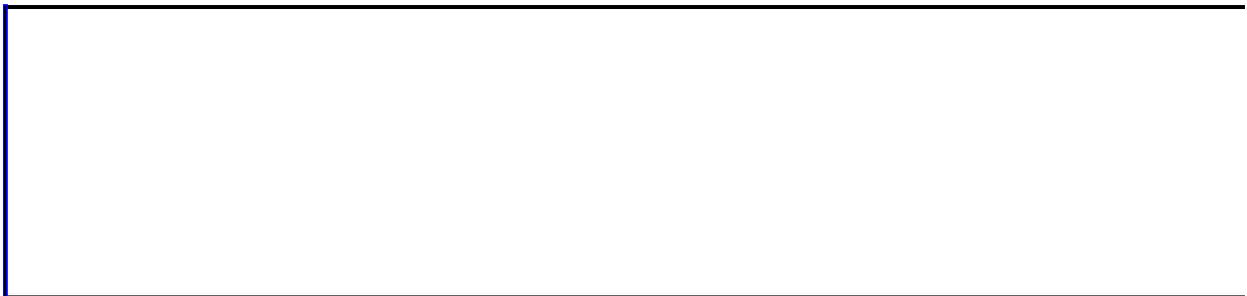
*expression* Required. An expression that returns a **Range** object.

## Example

[As it applies to the \*\*Range\*\* object.](#)

In this example, Microsoft Excel notifies the user if cell A1 can be edited or not on a protected worksheet.

```
Sub UseAllowEdit()  
    Dim wksOne As Worksheet  
  
    Set wksOne = Application.ActiveSheet  
  
    ' Protect the worksheet  
    wksOne.Protect  
  
    ' Notify the user about editing cell A1.  
    If wksOne.Range("A1").AllowEdit = True Then  
        MsgBox "Cell A1 can be edited."  
    Else  
        MsgBox "Cell A1 cannot be edited."  
    End If  
  
End Sub
```



# AllowEditRanges Property

Returns an [AllowEditRanges](#) object.

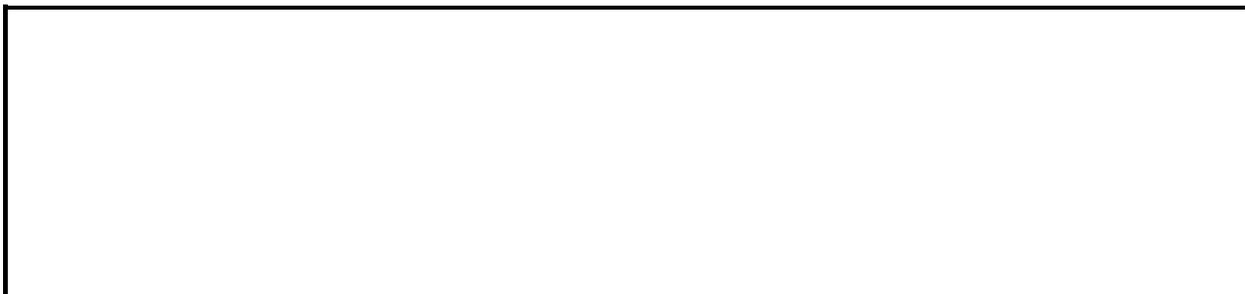
*expression*.**AllowEditRanges**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel allows edits to range A1:A4 on the active worksheet and notifies the user of the title and address of the specified range.

```
Sub UseAllowEditRanges()  
  
    Dim wksOne As Worksheet  
    Dim strPwd1 As String  
  
    Set wksOne = Application.ActiveSheet  
  
    strPwd1 = InputBox("Enter Password")  
  
    ' Unprotect worksheet.  
    wksOne.Unprotect  
  
    ' Establish a range that can allow edits  
    ' on the protected worksheet.  
    wksOne.Protection.AllowEditRanges.Add _  
        Title:="Classified", _  
        Range:=Range("A1:A4"), _  
        Password:=strPwd1  
  
    ' Notify the user  
    ' the title and address of the range.  
    With wksOne.Protection.AllowEditRanges.Item(1)  
        MsgBox "Title of range: " & .Title  
        MsgBox "Address of range: " & .Range.Address  
    End With  
  
End Sub
```



[Show All](#)

# AllowFillIn Property

Returns a **Boolean** value indicating whether users can provide their own data for cells in a column (rather than being restricted to a list of values) for those columns that supply a list of values. Returns **False** for [lists](#) that are not linked to a SharePoint site. Also returns **False** if the column is not specified as choice or multi-choice. Read-only **Boolean**.

*expression*.**AllowFillIn**

*expression* Required. An expression that returns one of the objects in the Applies To list.



# AllowFiltering Property

Returns **True** if the user is allowed to make use of an AutoFilter that was created before the sheet was protected. Read-only **Boolean**.

*expression*.**AllowFiltering**

*expression* Required. An expression that returns a [Protection](#) object.

## Remarks

The **AllowFiltering** property can be set by using the [Protect](#) method arguments.

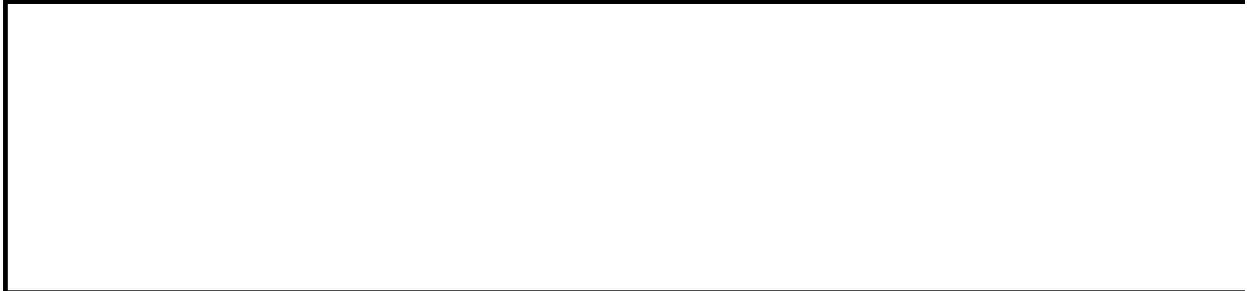
The **AllowFiltering** property allows the user to change filter criteria on an existing AutoFilter. The user cannot create or remove an AutoFilter on a protected worksheet.

The cells to be filtered must be unlocked when the sheet is protected.

## Example

This example allows the user to filter row 1 on the protected worksheet and notifies the user.

```
Sub ProtectionOptions()  
    ActiveSheet.Unprotect  
  
    ' Unlock row 1.  
    Rows("1:1").Locked = False  
  
    ' Allow row 1 to be filtered on a protected worksheet.  
    If ActiveSheet.Protection.AllowFiltering = False Then  
        ActiveSheet.Protect AllowFiltering:=True  
    End If  
  
    MsgBox "Row 1 can be filtered on this protected worksheet."  
  
End Sub
```



# AllowFormattingCells Property

Returns **True** if the formatting of cells is allowed on a protected worksheet.  
Read-only **Boolean**.

*expression*.**AllowFormattingCells**

*expression* Required. An expression that returns a [Protection](#) object.

## Remarks

The **AllowFormattingCells** property can be set by using the [Protect](#) method arguments.

Use of this property disables the protection tab, allowing the user to change all formats, but not to unlock or unhide ranges.

## Example

This example allows the user to format cells on the protected worksheet and notifies the user.

```
Sub ProtectionOptions()  
    ActiveSheet.Unprotect  
  
    ' Allow cells to be formatted on a protected worksheet.  
    If ActiveSheet.Protection.AllowFormattingCells = False Then  
        ActiveSheet.Protect AllowFormattingCells:=True  
    End If  
  
    MsgBox "Cells can be formatted on this protected worksheet."  
  
End Sub
```



# AllowFormattingColumns Property

Returns **True** if the formatting of columns is allowed on a protected worksheet.  
Read-only **Boolean**.

*expression*.**AllowFormattingColumns**

*expression* Required. An expression that returns a [Protection](#) object.

## Remarks

The **AllowFormattingColumns** property can be set by using the [Protect](#) method arguments.

## Example

This example allows the user to format columns on the protected worksheet and notifies the user.

```
Sub ProtectionOptions()  
    ActiveSheet.Unprotect  
  
    ' Allow columns to be formatted on a protected worksheet.  
    If ActiveSheet.Protection.AllowFormattingColumns = False Then  
        ActiveSheet.Protect AllowFormattingColumns:=True  
    End If  
  
    MsgBox "Columns can be formatted on this protected worksheet."  
End Sub
```



# AllowFormattingRows Property

Returns **True** if the formatting of rows is allowed on a protected worksheet.  
Read-only **Boolean**.

*expression*.**AllowFormattingRows**

*expression* Required. An expression that returns a [Protection](#) object.

## Remarks

The **AllowFormattingRows** property can be set by using the [Protect](#) method arguments.

## Example

This example allows the user to format the rows on the protected worksheet and notifies the user.

```
Sub ProtectionOptions()  
    ActiveSheet.Unprotect  
  
    ' Allow rows to be formatted on a protected worksheet.  
    If ActiveSheet.Protection.AllowFormattingRows = False Then  
        ActiveSheet.Protect AllowFormattingRows:=True  
    End If  
  
    MsgBox "Rows can be formatted on this protected worksheet."  
  
End Sub
```



# AllowInsertingColumns Property

Returns **True** if the insertion of columns is allowed on a protected worksheet.  
Read-only **Boolean**.

*expression*.**AllowInsertingColumns**

*expression* Required. An expression that returns a [Protection](#) object.

## Remarks

An inserted column inherits its formatting (by default) from the column to its left, which means that it may have locked cells. In other words, users may not be able to delete columns that they have inserted.

The **AllowInsertingColumns** property can be set by using the [Protect](#) method arguments.

## Example

This example allows the user to insert columns on the protected worksheet and notifies the user.

```
Sub ProtectionOptions()  
    ActiveSheet.Unprotect  
  
    ' Allow columns to be inserted on a protected worksheet.  
    If ActiveSheet.Protection.AllowInsertingColumns = False Then  
        ActiveSheet.Protect AllowInsertingColumns:=True  
    End If  
  
    MsgBox "Columns can be inserted on this protected worksheet."  
  
End Sub
```



# AllowInsertingHyperlinks Property

Returns **True** if the insertion of hyperlinks is allowed on a protected worksheet.  
Read-only **Boolean**.

*expression*.**AllowInsertingHyperlinks**

*expression* Required. An expression that returns a [Protection](#) object.

## Remarks

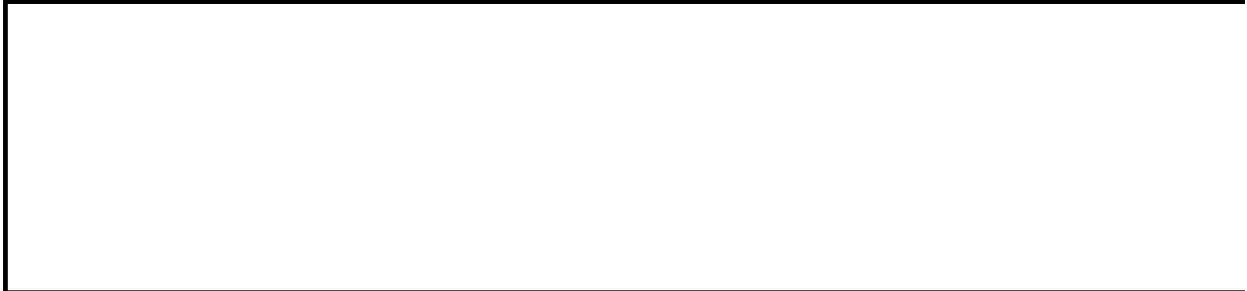
Hyperlinks can only be inserted in unlocked or unprotected cells on a protected worksheet.

The **AllowInsertingHyperlinks** property can be set by using the [Protect](#) method arguments.

## Example

This example allows the user to insert a hyperlink in cell A1 on the protected worksheet and notifies the user.

```
Sub ProtectionOptions()  
    ActiveSheet.Unprotect  
  
    ' Unlock cell A1.  
    Range("A1").Locked = False  
  
    ' Allow hyperlinks to be inserted on a protected worksheet.  
    If ActiveSheet.Protection.AllowInsertingHyperlinks = False Then  
        ActiveSheet.Protect AllowInsertingHyperlinks:=True  
    End If  
  
    MsgBox "Hyperlinks can be inserted on this protected worksheet."  
End Sub
```



# AllowInsertingRows Property

Returns **True** if the insertion of rows is allowed on a protected worksheet. Read-only **Boolean**.

*expression*.**AllowInsertingRows**

*expression* Required. An expression that returns a [Protection](#) object.

## Remarks

The **AllowInsertingRows** property can be set by using the [Protect](#) method arguments.

## Example

This example allows the user to insert rows on the protected worksheet and notifies the user.

```
Sub ProtectionOptions()  
    ActiveSheet.Unprotect  
  
    ' Allow rows to be inserted on a protected worksheet.  
    If ActiveSheet.Protection.AllowInsertingRows = False Then  
        ActiveSheet.Protect AllowInsertingRows:=True  
    End If  
  
    MsgBox "Rows can be inserted on this protected worksheet."  
  
End Sub
```



# AllowPNG Property

**True** if PNG (Portable Network Graphics) is allowed as an image format when you save documents as a Web page. **False** if PNG is not allowed as an output format. The default value is **False**. Read/write **Boolean**.

## Remarks

If you save images in the PNG format as opposed to any other file format, you might improve the image quality or reduce the size of those image files, and therefore decrease the download time, assuming that the Web browsers you are targeting support the PNG format.

## Example

This example enables PNG as an output format for the first workbook.

```
Application.Workbooks(1).WebOptions.AllowPNG = True
```

Alternatively, PNG can be enabled as the global default for the application for newly created documents.

```
Application.DefaultWebOptions.AllowPNG = True
```



# AllowSorting Property

Returns **True** if the sorting option is allowed on a protected worksheet. Read-only **Boolean**.

*expression*.**AllowSorting**

*expression* Required. An expression that returns a [Protection](#) object.

## Remarks

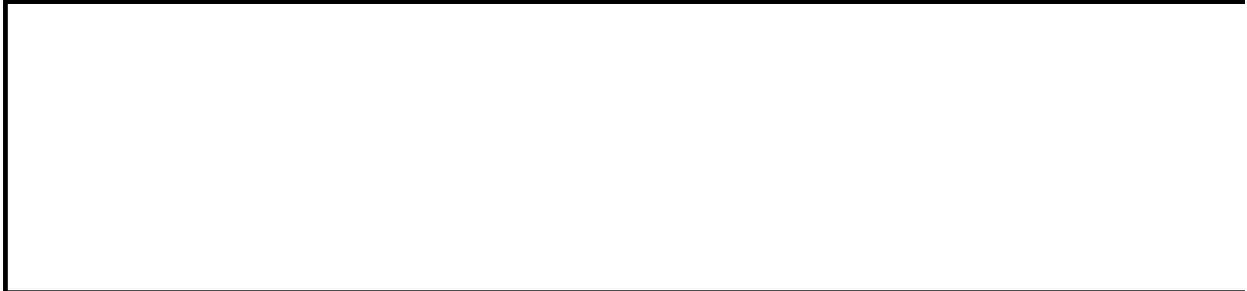
Sorting can only be performed on unlocked or unprotected cells in a protected worksheet.

The **AllowSorting** property can be set by using the [Protect](#) method arguments.

## Example

This example allows the user to sort unlocked or unprotected cells on the protected worksheet and notifies the user.

```
Sub ProtectionOptions()  
    ActiveSheet.Unprotect  
  
    ' Unlock cells A1 through B5.  
    Range("A1:B5").Locked = False  
  
    ' Allow sorting to be performed on the protected worksheet.  
    If ActiveSheet.Protection.AllowSorting = False Then  
        ActiveSheet.Protect AllowSorting:=True  
    End If  
  
    MsgBox "For cells A1 through B5, sorting can be performed on the  
End Sub
```



[Show All](#)

# AllowUsingPivotTables Property

Returns **True** if the user is allowed to manipulate pivot tables on a protected worksheet. Read-only **Boolean**.

*expression*.**AllowUsingPivotTables**

*expression* Required. An expression that returns a [Protection](#) object.

## Remarks

The **AllowUsingPivotTables** property applies to [non-OLAP source data](#).

The **AllowUsingPivotTables** property can be set by using the [Protect](#) method arguments.

## Example

This example allows the user to access the PivotTable report and notifies the user. It assumes a non-OLAP Pivot Table report exists on the active worksheet.

```
Sub ProtectionOptions()  
    ActiveSheet.Unprotect  
  
    ' Allow pivot tables to be manipulated on a protected worksheet.  
    If ActiveSheet.Protection.Allow UsingPivotTables = False Then  
        ActiveSheet.Protect AllowUsingPivotTables:=True  
    End If  
  
    MsgBox "Pivot tables can be manipulated on the protected workshe  
End Sub
```



# AlternativeText Property

Returns or sets the descriptive (alternative) text string for a [Shape](#) or [ShapeRange](#) object when the object is saved to a Web page. Read/write **String**.

## Remarks

The alternative text can be displayed either in place of the shape's image in the Web browser , or directly over the image when the mouse pointer hovers over the image (in browsers that support these features).

## Example

This example sets the alternative text for the first shape on the first worksheet to a description of the shape.

```
Worksheets(1).Shapes(1).AlternativeText = "Concentric circles"
```



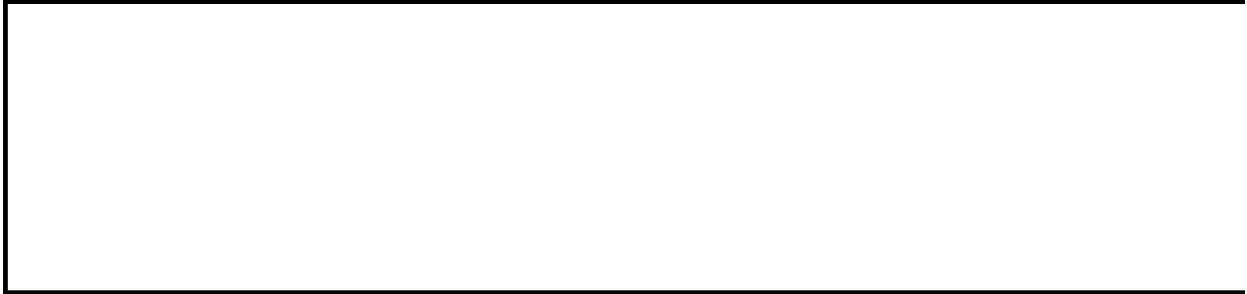
# AltStartupPath Property

Returns or sets the name of the alternate startup folder. Read/write **String**.

## Example

This example sets the alternate startup folder.

```
Application.AltStartupPath = "C:\EXCEL\MACROS"
```



# AlwaysSaveInDefaultEncoding Property

**True** if the default encoding is used when you save a Web page or plain text document, independent of the file's original encoding when opened. **False** if the original encoding of the file is used. The default value is **False**. Read/write **Boolean**.

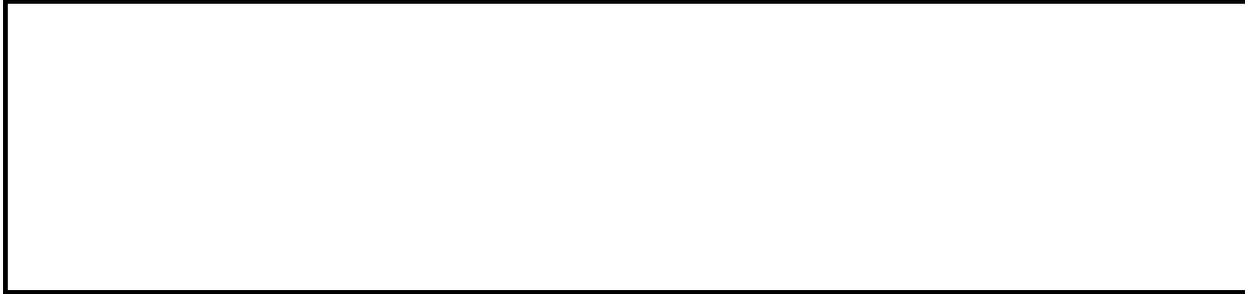
## Remarks

The [Encoding](#) property can be used to set the default encoding.

## Example

This example sets the encoding to the default encoding. The encoding is used when you save the document as a Web page.

```
Application.DefaultWebOptions.AlwaysSaveInDefaultEncoding = True
```



[Show All](#)

# Angle Property

Returns or sets the angle of the callout line. If the callout line contains more than one line segment, this property returns or sets the angle of the segment that is farthest from the callout text box. Read/write [MsoCalloutAngleType](#).

MsoCalloutAngleType can be one of these MsoCalloutAngleType constants.

**msoCalloutAngle30**

**msoCalloutAngle45**

**msoCalloutAngle60**

**msoCalloutAngle90**

**msoCalloutAngleAutomatic**

**msoCalloutAngleMixed**

*expression*.**Angle**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

If you set the value of this property to anything other than **msoCalloutAngleAutomatic**, the callout line maintains a fixed angle as you drag the callout.

## Example

This example sets to 90 degrees the callout angle for a callout named "callout1" on myDocument.

```
Set myDocument = Worksheets(1)  
myDocument.Shapes("callout1").Callout.Angle = msoCalloutAngle90
```



# AnswerWizard Property

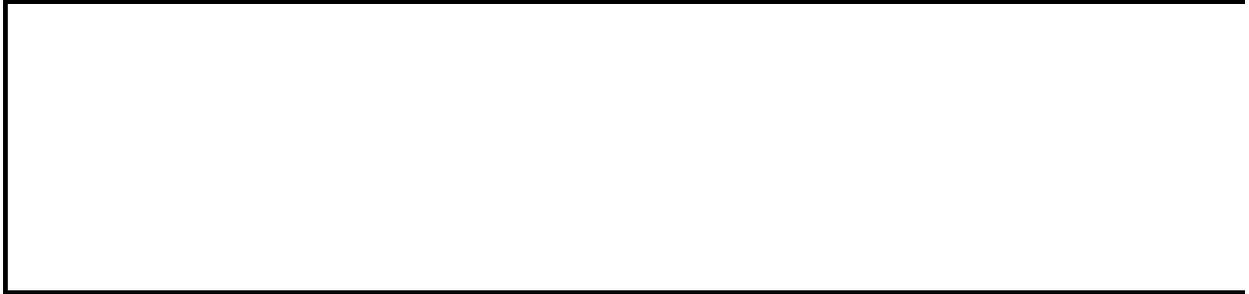
Some of the content in this topic may not be applicable to some languages.

Returns the [AnswerWizard](#) object for Microsoft Excel. Read-only.

## Example

This example resets the Answer Wizard file list.

```
Application.AnswerWizard.ResetFileList
```



# AppendOnImport Property

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

**True** if you want to append new rows to XML lists that are bound to the specified schema map when you are importing new data or refreshing an existing connection. **False** if you want to overwrite the contents of cells that are bound to the specified schema map when you are importing new data or refreshing an existing connection. The default value is **False**. Read/write **Boolean**.

*expression*.**AppendOnImport**

*expression* Required. An expression that returns an [XmlMap](#) object.



# Application Property

When used *without* an object qualifier, this property returns an [Application](#) object that represents the Microsoft Excel application. When used *with* an object qualifier, this property returns an **Application** object that represents the creator of the specified object (you can use this property with an OLE Automation object to return the application of that object). Read-only.

*expression*.**Application**

*expression* Required. An expression that returns one of the above objects.

## Example

This example displays a message about the application that created myObject.

```
Set myObject = ActiveWorkbook
If myObject.Application.Value = "Microsoft Excel" Then
    MsgBox "This is an Excel Application object."
Else
    MsgBox "This is not an Excel Application object."
End If
```



# ApplyPictToEnd Property

**True** if a picture is applied to the end of the point or all points in the series.  
Read/write **Boolean**.

## Example

This example applies pictures to the end of all points in series one. The series must already have pictures applied to it (setting this property changes the picture orientation).

```
Charts(1).SeriesCollection(1).ApplyPictToEnd = True
```



# ApplyPictToFront Property

**True** if a picture is applied to the front of the point or all points in the series.  
Read/write **Boolean**.

## Example

This example applies pictures to the front of all points in series one. The series must already have pictures applied to it (setting this property changes the picture orientation).

```
Charts(1).SeriesCollection(1).ApplyPictToFront = True
```



# ApplyPictToSides Property

**True** if a picture is applied to the sides of the point or all points in the series.  
Read/write **Boolean**.

## Example

This example applies pictures to the sides of all points in series one. The series must already have pictures applied to it (setting this property changes the picture orientation).

```
Charts(1).SeriesCollection(1).ApplyPictToSides = True
```



[Show All](#)

# ArabicModes Property

Returns or sets the mode for the Arabic spelling checker. Read/write [XlArabicModes](#).

XlArabicModes can be one of these XlArabicModes constants.

**xlArabicNone** The spelling checker ignores spelling rules regarding either Arabic words ending with the letter yaa or Arabic words beginning with an alef hamza.

**xlArabicBothStrict** The spelling checker uses spelling rules regarding both Arabic words ending with the letter yaa and Arabic words beginning with an alef hamza.

**xlArabicStrictAlefHamza** The spelling checker uses spelling rules regarding Arabic words beginning with an alef hamza.

**xlArabicStrictFinalYaa** The spelling checker uses spelling rules regarding Arabic words ending with the letter yaa.

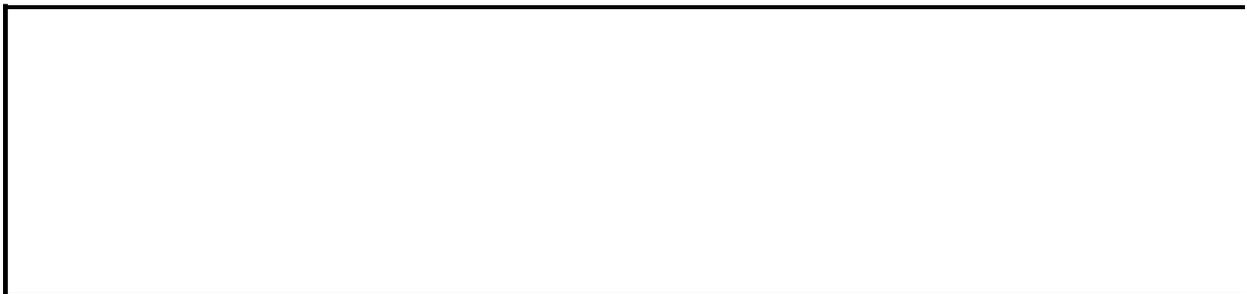
*expression*.**ArabicModes**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel checks the setting for the spell checking option for Arabic mode and sets it to check for words ending with the letter yaa and words beginning with an alef hamza, if the Arabic mode is not set to this already. Before running this code example, the Arabic modes option must be enabled in the spelling options.

```
Sub SpellCheck()  
  
    If Application.SpellingOptions.ArabicModes <> xlArabicBothStrict  
        Application.SpellingOptions.ArabicModes = xlArabicBothStrict  
        MsgBox "Spell checking for Arabic mode has been changed to a  
    Else  
        MsgBox "Spell checking for Arabic mode is already in a stric  
    End If  
  
End Sub
```



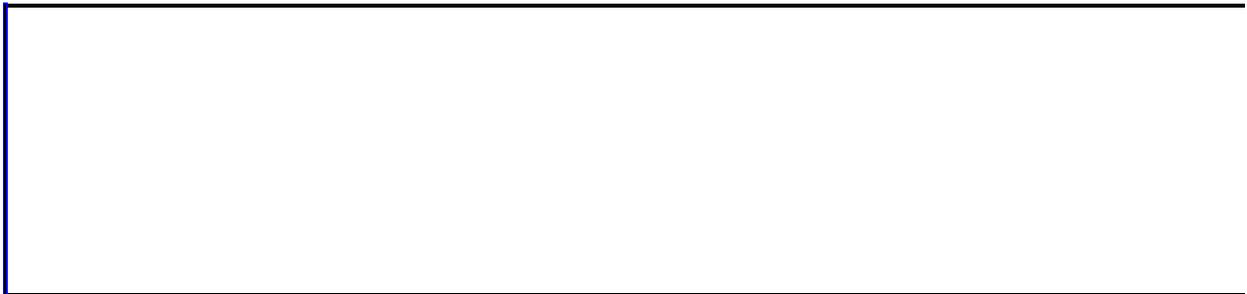
# ArbitraryXMLSupportAvailable Property

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Returns a **Boolean** value that indicates whether the XML features in Microsoft Excel are available. Read-only.

*expression*.**ArbitraryXMLSupportAvailable**

*expression* Required. An expression that returns an [Application](#) object.



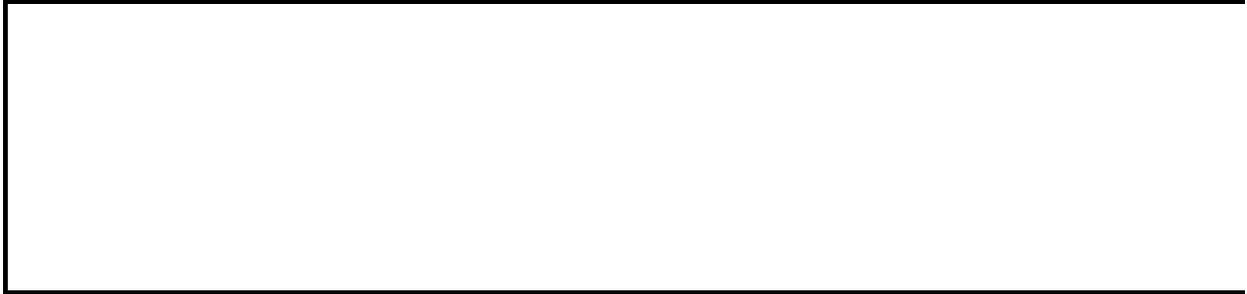
# Area3DGroup Property

Returns a [ChartGroup](#) object that represents the area chart group on a 3-D chart. Read-only.

## Example

This example turns on drop lines for the 3-D area chart group.

```
Charts(1).Area3DGroup.HasDropLines = True
```



# Areas Property

Returns an [Areas](#) collection that represents all the ranges in a multiple-area selection. Read-only.

For information about returning a single member of a collection, see [Returning an Object from a Collection](#).

## Remarks

For a single selection, the **Areas** property returns a collection that contains one object—the original **Range** object itself. For a multiple-area selection, the **Areas** property returns a collection that contains one object for each selected area.

## Example

This example displays a message if the user tries to carry out a command when more than one area is selected. This example must be run from a worksheet.

```
If Selection.Areas.Count > 1 Then  
    MsgBox "Cannot do this to a multi-area selection."  
End If
```



# AskToUpdateLinks Property

**True** if Microsoft Excel asks the user to update links when opening files with links. **False** if links are automatically updated with no dialog box. Read/write **Boolean**.

## Example

This example sets Microsoft Excel to ask the user to update links whenever a file that contains links is opened.

```
Application.AskToUpdateLinks = True
```



# Assistant Property

Some of the content in this topic may not be applicable to some languages.

Returns an [Assistant](#) object for Microsoft Excel.

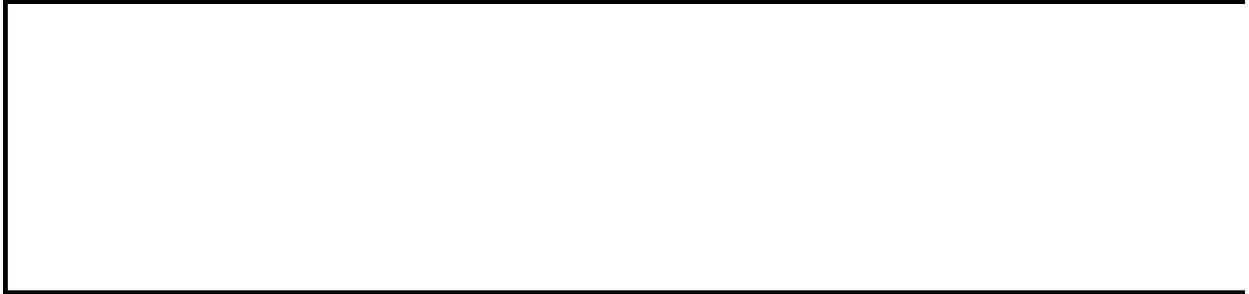
## Remarks

Using this property without an object qualifier is equivalent to using `Application.Assistant`.

## Example

This example makes the Office Assistant visible.

```
Assistant.Visible = True
```



# Author Property

Returns or sets the author of the comment. Read-only **String**.

## Example

This example deletes all comments added by Jean Selva on the active sheet.

```
For Each c in ActiveSheet.Comments
    If c.Author = "Jean Selva" Then c.Delete
Next
```



[Show All](#)

# AutoAttach Property

**True** if the place where the callout line attaches to the callout text box changes depending on whether the origin of the callout line (where the callout points to) is to the left or right of the callout text box. Read/write [MsoTriState](#).

MsoTriState can be one of these MsoTriState constants.

**msoCTrue**

**msoFalse**

**msoTriStateMixed**

**msoTriStateToggle**

**msoTrue** The place where the callout line attaches to the callout text box changes depending on whether the origin of the callout line (where the callout points to) is to the left or right of the callout text box.

## Remarks

When the value of this property is **True**, the drop value (the vertical distance from the edge of the callout text box to the place where the callout line attaches) is measured from the top of the text box when the text box is to the right of the origin, and it's measured from the bottom of the text box when the text box is to the left of the origin. When the value of this property is **False**, the drop value is always measured from the top of the text box, regardless of the relative positions of the text box and the origin. Use the [CustomDrop](#) method to set the drop value, and use the [Drop](#) property to return the drop value.

Setting this property affects a callout only if it has an explicitly set drop value — that is, if the value of the [DropType](#) property is **msoCalloutDropCustom**. By default, callouts have explicitly set drop values when they're created.

## Example

This example adds two callouts to myDocument. If you drag the text box for each of these callouts to the left of the callout line origin, the place on the text box where the callout line attaches will change for the automatically attached callout.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes
    With .AddCallout(msoCalloutTwo, 420, 170, 200, 50)
        .TextFrame.Characters.Text = "auto-attached"
        .Callout.AutoAttach = True
    End With
    With .AddCallout(msoCalloutTwo, 420, 350, 200, 50)
        .TextFrame.Characters.Text = "not auto-attached"
        .Callout.AutoAttach = False
    End With
End With
```



# AutoCorrect Property

Returns an [AutoCorrect](#) object that represents the Microsoft Excel AutoCorrect attributes. Read-only.

## Example

This example substitutes the word "Temp." for the word "Temperature" in the array of AutoCorrect replacements.

```
With Application.AutoCorrect  
    .AddReplacement "Temperature", "Temp."  
End With
```



[Show All](#)

# AutoExpandListRange Property

A **Boolean** value indicating whether automatic expansion is enabled for [lists](#). When you type in a cell of an empty row or column next to a list, the list will expand to include that row or column if automatic expansion is enabled.

Read/write **Boolean**.

*expression*.**AutoExpandListRange**

*expression* Required. An expression that returns one of the objects in the Applies To list.

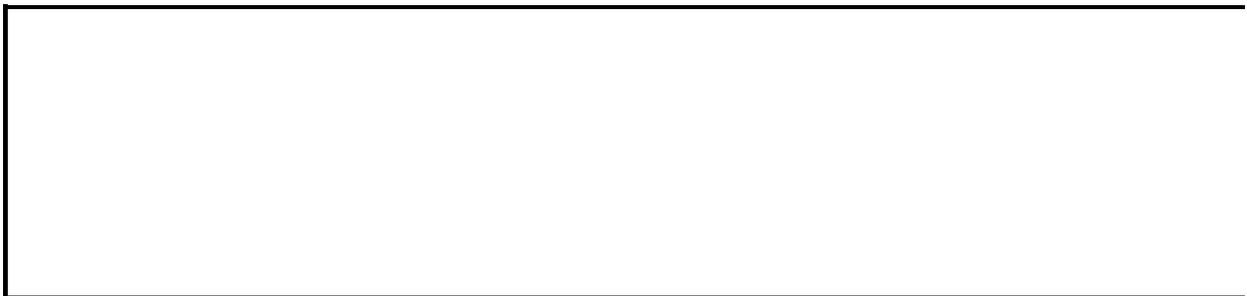
## Example

The following example enables automatic expansion of lists when typing in adjacent rows or columns.

```
Sub SetAutoExpand
```

```
    Application.AutoCorrect.AutoExpandListRange = TRUE
```

```
End Sub
```



# AutoFilter Property

Returns an [AutoFilter](#) object if filtering is on. Returns **Nothing** if filtering is off.  
Read-only.

## Remarks

To create an **AutoFilter** object for a worksheet, you must turn autofiltering on for a range on the worksheet either manually or using the **AutoFilter** method of the **Range** object.

## Example

The following example sets a variable to the value of the **Criteria1** property of the filter for the first column in the filtered range on the Crew worksheet.

```
With Worksheets("Crew")
    If .AutoFilterMode Then
        With .AutoFilter.Filters(1)
            If .On Then c1 = .Criteria1
        End With
    End If
End With
```



# AutoFilterMode Property

**True** if the AutoFilter drop-down arrows are currently displayed on the sheet. This property is independent of the **FilterMode** property. Read/write **Boolean**.

## Remarks

This property returns **True** if the drop-down arrows are currently displayed. You can set this property to **False** to remove the arrows, but you cannot set it to **True**. Use the [AutoFilter](#) method to filter a list and display the drop-down arrows.

## Example

This example displays the current status of the **AutoFilterMode** property on Sheet1.

```
If Worksheets("Sheet1").AutoFilterMode Then
    isOn = "On"
Else
    isOn = "Off"
End If
MsgBox "AutoFilterMode is " & isOn
```



[Show All](#)

# AutoFormat Property

Sets or returns an [MsoTriState](#) constant indicating the automatic formatting state for a diagram. Read/write.

MsoTriState can be one of these MsoTriState constants.

**msoCTrue** Does not apply to this property.

**msoFalse** Disables automatic formatting for the diagram.

**msoTriStateMixed** Does not apply to this property.

**msoTriStateToggle** Does not apply to this property.

**msoTrue** Enables automatic formatting for the diagram.

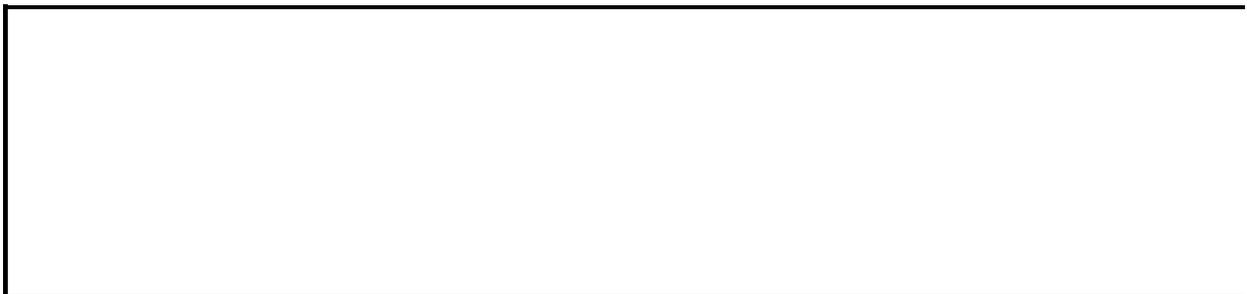
*expression*.**AutoFormat**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example creates a diagram in the current document and turns on the automatic format for the diagram.

```
Sub CreatePyramidDiagram()  
    Dim dgnNode As DiagramNode  
    Dim shpDiagram As Shape  
    Dim intCount As Integer  
  
    'Add a pyramid diagram to current document.  
    Set shpDiagram = ActiveSheet.Shapes.AddDiagram( _  
        Type:=msoDiagramPyramid, _  
        Left:=10, _  
        Top:=15, _  
        Width:=400, _  
        Height:=475)  
  
    'Add first child node.  
    Set dgnNode = shpDiagram.DiagramNode.Children.AddNode  
  
    'Add three more child nodes  
    For intCount = 1 To 3  
        dgnNode.AddNode  
    Next intCount  
  
    'Enable automatic formatting for the diagram and convert  
    'it to a radial diagram.  
    With dgnNode.Diagram  
        .AutoFormat = msoTrue  
        .Convert Type:=msoDiagramRadial  
    End With  
  
End Sub
```



# AutoFormatAsYouTypeReplaceHyperProperty

**True** (default) if Microsoft Excel automatically formats hyperlinks as you type.  
**False** if Excel does not automatically format hyperlinks as you type. Read/write **Boolean**.

*expression*.**AutoFormatAsYouTypeReplaceHyperlinks**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel determines if the ability to format hyperlinks automatically as they are typed is enabled and notifies the user.

```
Sub CheckHyperlinks()  
    ' Determine if automatic formatting is enabled and notify user.  
    If Application.AutoFormatAsYouTypeReplaceHyperlinks = True Then  
        MsgBox "Automatic formatting for typing in hyperlinks is ena  
    Else  
        MsgBox "Automatic formatting for typing in hyperlinks is not  
    End If  
End Sub
```



[Show All](#)

# AutoLayout Property

Returns or sets an [MsoTriState](#) constant which determines the automatic positioning of the nodes and connectors in a diagram. Read/write.

MsoTriState can be one of these MsoTriState constants.

**msoCTrue** Does not apply to this property.

**msoFalse** Disables automatic formatting for the diagram.

**msoTriStateMixed** Does not apply to this property.

**msoTriStateToggle** Does not apply to this property.

**msoTrue** Enables automatic formatting for the diagram.

*expression*.**AutoLayout**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example creates a diagram in the current document and automatically positions the nodes and connectors.

```
Sub CreatePyramidDiagram()  
    Dim dgnNode As DiagramNode  
    Dim shpDiagram As Shape  
    Dim intCount As Integer  
  
    Set shpDiagram = ActiveSheet.Shapes.AddDiagram( _  
        Type:=msoDiagramPyramid, _  
        Left:=10, _  
        Top:=15, _  
        Width:=400, _  
        Height:=475)  
    Set dgnNode = shpDiagram.DiagramNode.Children.AddNode  
  
    For intCount = 1 To 3  
        dgnNode.AddNode  
    Next intCount  
  
    With dgnNode.Diagram  
        .AutoLayout = msoTrue  
        .Convert Type:=msoDiagramRadial  
    End With  
  
End Sub
```



[Show All](#)

# AutoLength Property

Applies only to callouts whose lines consist of more than one segment (types **msoCalloutThree** and **msoCalloutFour**). Read/write [MsoTriState](#).

MsoTriState can be one of these MsoTriState constants.

**msoCTrue**

**msoFalse** The first segment of the callout retains the fixed length specified by the [Length](#) property whenever the callout is moved.

**msoTriStateMixed**

**msoTriStateToggle**

**msoTrue** The first segment of the callout line (the segment attached to the text callout box) is scaled automatically whenever the callout is moved.

## Remarks

This property is read-only. Use the [AutomaticLength](#) method to set this property to **msbTrue**, and use the [CustomLength](#) method to set this property to **msbFalse**.

## Example

This example toggles between an automatically scaling first segment and one with a fixed length for the callout line for shape one on myDocument. For the example to work, shape one must be a callout.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(1).Callout
    If .AutoLength Then
        .CustomLength 50
    Else
        .AutomaticLength
    End If
End With
```



# AutoLoad Property

**True** if the OLE object is automatically loaded when the workbook that contains it is opened. Read/write **Boolean**.

## Remarks

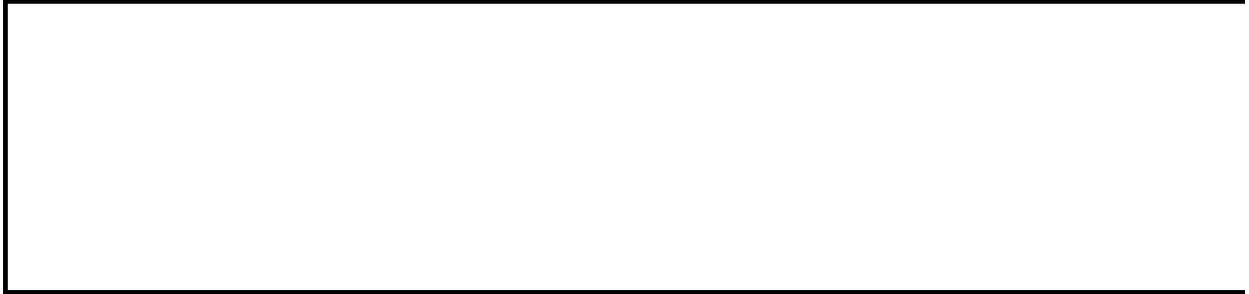
This property is ignored by ActiveX controls. ActiveX controls are always loaded when a workbook is opened.

For most OLE object types, this property shouldn't be set to **True**. By default, the **AutoLoad** property is set to **False** for new OLE objects; this saves time and memory when Microsoft Excel is loading workbooks. The benefit of automatically loading OLE objects is that, for objects that represent volatile data, links to source data can be reestablished immediately and the objects can be rendered again, if necessary.

## Example

This example sets the **AutoLoad** property for OLE object one on the active sheet.

```
ActiveSheet.OLEObjects(1).AutoLoad = False
```



# AutoMargins Property

**True** if Microsoft Excel automatically calculates text frame margins. Read/write **Boolean**.

## Remarks

When this property is **True**, the **MarginLeft**, **MarginRight**, **MarginTop**, and **MarginBottom** properties are ignored.

## Example

This example causes Microsoft Excel to automatically calculate text frame margins for text in shape one.

```
Worksheets(1).Shapes(1).TextFrame.AutoMargins = True
```



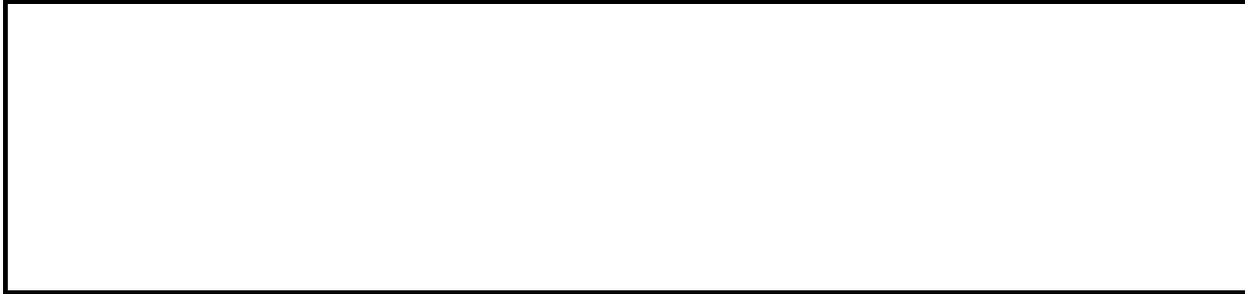
# AutomaticStyles Property

**True** if the outline uses automatic styles. Read/write **Boolean**.

## Example

This example sets the outline on Sheet1 to use automatic styles.

```
Worksheets("Sheet1").Outline.AutomaticStyles = True
```



[Show All](#)

# AutomationSecurity Property

Returns or sets an [MsoAutomationSecurity](#) constant that represents the security mode Microsoft Excel uses when programmatically opening files. This property is automatically set to **msoAutomationSecurityLow** when the application is started. Therefore, to avoid breaking solutions that rely on the default setting, you should be careful to reset this property to **msoAutomationSecurityLow** after programmatically opening a file. Also, this property should be set immediately before and after opening a file programmatically to avoid malicious subversion. Read/write.

MsoAutomationSecurity can be one of these MsoAutomationSecurity constants.

**msoAutomationSecurityByUI** Uses the security setting specified in the **Security** dialog box.

**msoAutomationSecurityForceDisable** Disables all macros in all files opened programmatically without showing any security alerts.

**msoAutomationSecurityLow** Enables all macros. This is the default value when the application is started.

*expression*.**AutomationSecurity**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Setting [ScreenUpdating](#) to **False** does not affect alerts and will not affect security warnings. The [DisplayAlerts](#) setting will not apply to security warnings. For example, if the user sets **DisplayAlerts** equal to **False** and **AutomationSecurity** to **msoAutomationSecurityByUI**, while the user is on Medium security level, then there will be security warnings while the macro is running. This allows the macro to trap file open errors, while still showing the security warning if the file open succeeds.

## Example

This example captures the current automation security setting, changes the setting to disable macros, displays the **Open** dialog box, and after opening the selected document, sets the automation security back to its original setting.

```
Sub Security()  
    Dim secAutomation As MsoAutomationSecurity  
  
    secAutomation = Application.AutomationSecurity  
  
    Application.AutomationSecurity = msoAutomationSecurityForceDisab  
    Application.FileDialog(msoFileDialogOpen).Show  
  
    Application.AutomationSecurity = secAutomation  
End Sub
```



# AutoPercentEntry Property

**True** if entries in cells formatted as percentages aren't automatically multiplied by 100 as soon as they are entered. Read/write **Boolean**.

## Example

This example enables automatic multiplication by 100 for subsequent entries in cells formatted as percentages.

```
Application.AutoPercentEntry = False
```



# AutoRecover Property

Returns an [AutoRecover](#) object, which backs up all file formats on a timed interval.

*expression*.**AutoRecover**

*expression* Required. An expression that returns one of the objects in the Applies To list.

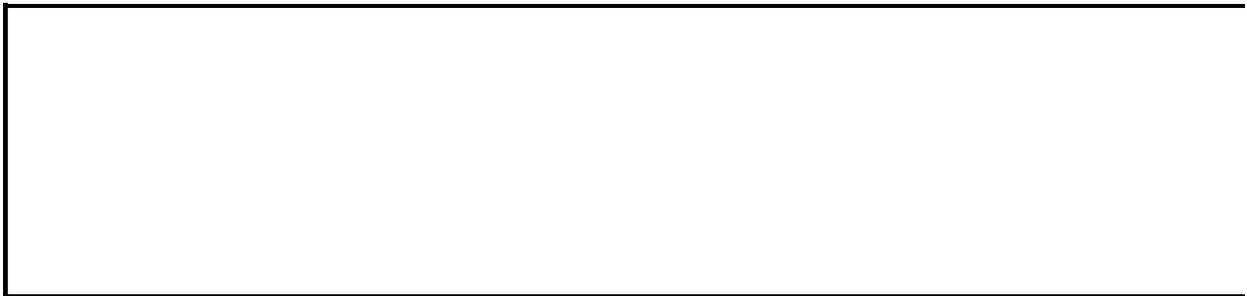
## **Remarks**

Valid time intervals are whole numbers from 1 to 120.

## Example

In this example, the **Time** property is used in conjunction with the **AutoRecover** property to set the time interval for Microsoft Excel to wait before saving another copy to five minutes.

```
Sub UseAutoRecover()  
    Application.AutoRecover.Time = 5  
    MsgBox "The time that will elapse between each automatic " & _  
        "save has been set to " & _  
        Application.AutoRecover.Time & " minutes."  
End Sub
```



# AutoRepublish Property

When a workbook is saved, Microsoft Excel determines if any item in the **PublishObjects** collection has the **AutoRepublish** property set to **True** and, if so, republishes it. The default value is **False**. Read/write **Boolean**.

*expression*.**AutoRepublish**

*expression* Required. An expression that returns a [PublishObject](#) object.

## Example

This example publishes a range on a worksheet to an HTML file on the C: drive. When the user saves the workbook containing the worksheet, Excel will automatically republish the range to the same HTML file. This example assumes that the user has read/write access to the web page and that cells A1 through D10 in the worksheet have values in them.

```
Sub PublishToWeb()  
  
    With ActiveWorkbook.PublishObjects.Add( _  
        SourceType:= xlSourceRange, _  
        Filename:="C:\Work.htm", _  
        Sheet:="Sheet1", _  
        Source:="A1:D10", _  
        HtmlType:=xlHtmlStatic, _  
        DivID:="Book1.xls_130489")  
        .Publish  
        .AutoRepublish = True  
    End With  
  
End Sub
```



# AutoScaleFont Property

**True** if the text in the object changes font size when the object size changes. The default value is **True**. Read/write **Variant**.

## Example

This example adds a title to embedded chart one on the active worksheet, and it causes the title font to remain the same size whenever the chart size changes.

```
With ActiveSheet.ChartObjects(1).Chart
    .HasTitle = True
    .ChartTitle.Text = "1996 sales"
    .ChartTitle.AutoScaleFont = False
End With
```



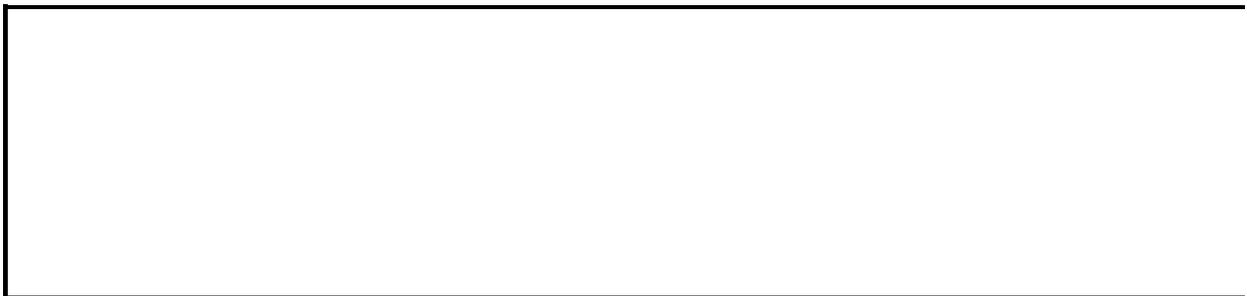
# AutoScaling Property

**True** if Microsoft Excel scales a 3-D chart so that it's closer in size to the equivalent 2-D chart. The [RightAngleAxes](#) property must be **True**. Read/write **Boolean**.

## Example

This example automatically scales Chart1. The example should be run on a 3-D chart.

```
With Charts("Chart1")  
    .RightAngleAxes = True  
    .AutoScaling = True  
End With
```



[Show All](#)

# AutoShapeType Property

Returns or sets the shape type for the specified **Shape** or **ShapeRange** object, which must represent an AutoShape other than a line, freeform drawing, or connector. Read/write [MsoAutoShapeType](#).

**Note** When you change the type of a shape, the shape retains its size, color, and other attributes.

MsoAutoShapeType can be one of these MsoAutoShapeType constants.

**msoShape24pointStar**

**msoShape4pointStar**

**msoShape8pointStar**

**msoShapeActionButtonBeginning**

**msoShapeActionButtonDocument**

**msoShapeActionButtonForwardorNext**

**msoShapeActionButtonHome**

**msoShapeActionButtonMovie**

**msoShapeActionButtonSound**

**msoShapeBalloon**

**msoShapeBentUpArrow**

**msoShapeBlockArc**

**msoShapeChevron**

**msoShapeCloudCallout**

**msoShapeCube**

**msoShapeCurvedDownRibbon**

**msoShapeCurvedRightArrow**

**msoShapeCurvedUpRibbon**

**msoShapeDonut**

**msoShapeDoubleBracket**

**msoShapeDownArrow**

**msoShapeDownRibbon**

**msoShapeExplosion2**  
**msoShapeFlowchartCard**  
**msoShapeFlowchartConnector**  
**msoShapeFlowchartDecision**  
**msoShapeFlowchartDirectAccessStorage**  
**msoShapeFlowchartDisplay**  
**msoShapeFlowchartDocument**  
**msoShapeFlowchartExtract**  
**msoShapeFlowchartInternalStorage**  
**msoShapeFlowchartMagneticDisk**  
**msoShapeFlowchartManualInput**  
**msoShapeFlowchartManualOperation**  
**msoShapeFlowchartMerge**  
**msoShapeFlowchartMultidocument**  
**msoShapeFlowchartOffpageConnector**  
**msoShapeFlowchartOr**  
**msoShapeFlowchartPredefinedProcess**  
**msoShapeFlowchartPreparation**  
**msoShapeFlowchartProcess**  
**msoShapeFlowchartPunchedTape**  
**msoShapeFlowchartSequentialAccessStorage**  
**msoShapeFlowchartSort**  
**msoShapeFlowchartStoredData**  
**msoShapeFlowchartSummingJunction**  
**msoShapeFlowchartTerminator**  
**msoShapeFoldedCorner**  
**msoShapeHeart**  
**msoShapeHexagon**  
**msoShapeHorizontalScroll**  
**msoShapeIsoscelesTriangle**  
**msoShapeLeftArrow**  
**msoShapeLeftArrowCallout**  
**msoShapeLeftBrace**

**msoShapeLeftBracket**  
**msoShapeLeftRightArrow**  
**msoShapeLeftRightArrowCallout**  
**msoShapeLeftRightUpArrow**  
**msoShapeLeftUpArrow**  
**msoShapeLightningBolt**  
**msoShapeLineCallout1**  
**msoShapeLineCallout1AccentBar**  
**msoShapeLineCallout1BorderandAccentBar**  
**msoShapeLineCallout1NoBorder**  
**msoShapeLineCallout2**  
**msoShapeLineCallout2AccentBar**  
**msoShapeLineCallout2BorderandAccentBar**  
**msoShapeLineCallout2NoBorder**  
**msoShapeLineCallout3**  
**msoShapeLineCallout3AccentBar**  
**msoShapeLineCallout3BorderandAccentBar**  
**msoShapeLineCallout3NoBorder**  
**msoShapeLineCallout4**  
**msoShapeLineCallout4AccentBar**  
**msoShapeLineCallout4BorderandAccentBar**  
**msoShapeLineCallout4NoBorder**  
**msoShapeMixed**  
**msoShapeMoon**  
**msoShapeNoSymbol**  
**msoShapeNotchedRightArrow**  
**msoShapeNotPrimitive**  
**msoShapeOctagon**  
**msoShapeOval**  
**msoShapeOvalCallout**  
**msoShapeParallelogram**  
**msoShapePentagon**  
**msoShapePlaque**

**msoShapeQuadArrowCallout**  
**msoShapeRectangularCallout**  
**msoShapeRightArrow**  
**msoShapeRightBrace**  
**msoShapeRightTriangle**  
**msoShapeRoundedRectangularCallout**  
**msoShapeStripedRightArrow**  
**msoShapeTrapezoid**  
**msoShapeUpArrowCallout**  
**msoShapeUpDownArrowCallout**  
**msoShapeUTurnArrow**  
**msoShapeWave**  
**msoShape16pointStar**  
**msoShape32pointStar**  
**msoShape5pointStar**  
**msoShapeActionButtonBackorPrevious**  
**msoShapeActionButtonCustom**  
**msoShapeActionButtonEnd**  
**msoShapeActionButtonHelp**  
**msoShapeActionButtonInformation**  
**msoShapeActionButtonReturn**  
**msoShapeArc**  
**msoShapeBentArrow**  
**msoShapeBevel**  
**msoShapeCan**  
**msoShapeCircularArrow**  
**msoShapeCross**  
**msoShapeCurvedDownArrow**  
**msoShapeCurvedLeftArrow**  
**msoShapeCurvedUpArrow**  
**msoShapeDiamond**  
**msoShapeDoubleBrace**  
**msoShapeDoubleWave**

**msoShapeDownArrowCallout**  
**msoShapeExplosion1**  
**msoShapeFlowchartAlternateProcess**  
**msoShapeFlowchartCollate**  
**msoShapeFlowchartData**  
**msoShapeFlowchartDelay**  
**msoShapeQuadArrow**  
**msoShapeRectangle**  
**msoShapeRegularPentagon**  
**msoShapeRightArrowCallout**  
**msoShapeRightBracket**  
**msoShapeRoundedRectangle**  
**msoShapeSmileyFace**  
**msoShapeSun**  
**msoShapeUpArrow**  
**msoShapeUpDownArrow**  
**msoShapeUpRibbon**  
**msoShapeVerticalScroll**

*expression*.**AutoShapeType**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Use the **Type** property of the [ConnectorFormat](#) object to set or return the connector type.

## Example

This example replaces all 16-point stars with 32-point stars in myDocument.

```
Set myDocument = Worksheets(1)
For Each s In myDocument.Shapes
    If s.AutoShapeType = msoShape16pointStar Then
        s.AutoShapeType = msoShape32pointStar
    End If
Next
```



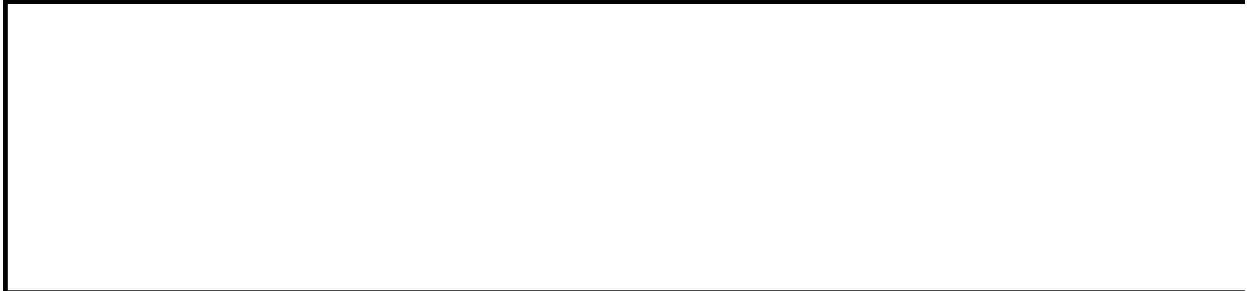
# AutoShowCount Property

Returns the number of top or bottom items that are automatically shown in the specified PivotTable field. Read-only **Long**.

## Example

This example displays a message box showing the **AutoShow** parameters for the Salesman field.

```
With Worksheets(1).PivotTables(1).PivotFields("salesman")
  If .AutoShowType = xlAutomatic Then
    r = .AutoShowRange
    If r = xlTop Then
      rn = "top"
    Else
      rn = "bottom"
    End If
    MsgBox "PivotTable report is showing " & rn & " " & _
      .AutoShowCount & " items in " & .Name & _
      " field by " & .AutoShowField
  Else
    MsgBox "PivotTable report is not using AutoShow for this fie
  End If
End With
```



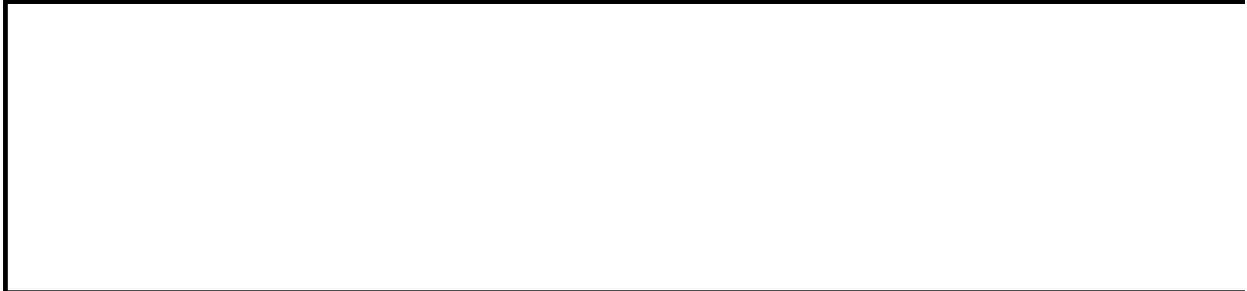
# AutoShowField Property

Returns the name of the data field used to determine the top or bottom items that are automatically shown in the specified PivotTable field. Read-only **String**.

## Example

This example displays a message box showing the **AutoShow** parameters for the Salesman field.

```
With Worksheets(1).PivotTables(1).PivotFields("salesman")
  If .AutoShowType = xlAutomatic Then
    r = .AutoShowRange
    If r = xlTop Then
      rn = "top"
    Else
      rn = "bottom"
    End If
    MsgBox "PivotTable report is showing " & rn & " " & _
      .AutoShowCount & " items in " & .Name & _
      " field by " & .AutoShowField
  Else
    MsgBox "PivotTable report is not using AutoShow for this fie
  End If
End With
```



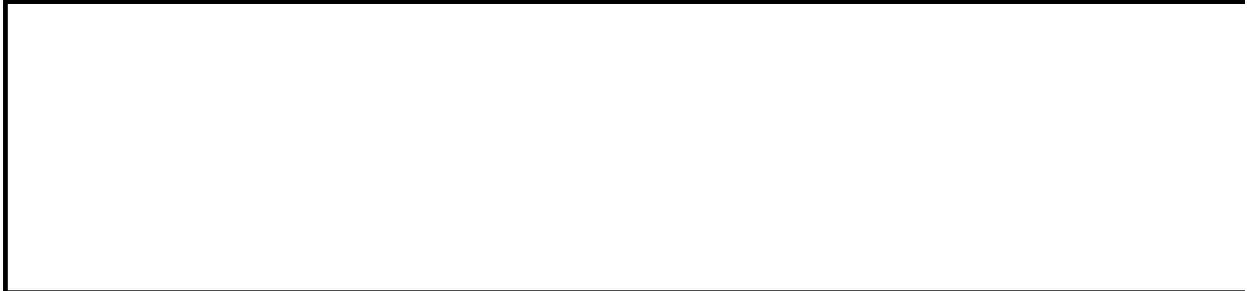
# AutoShowRange Property

Returns **xlTop** if the top items are shown automatically in the specified PivotTable field; returns **xlBottom** if the bottom items are shown. Read-only **Long**.

## Example

This example displays a message box showing the **AutoShow** parameters for the Salesman field.

```
With Worksheets(1).PivotTables(1).PivotFields("salesman")
    If .AutoShowType = xlAutomatic Then
        r = .AutoShowRange
        If r = xlTop Then
            rn = "top"
        Else
            rn = "bottom"
        End If
        MsgBox "PivotTable report is showing " & rn & " " & _
            .AutoShowCount & " items in " & .Name & _
            " field by " & .AutoShowField
    Else
        MsgBox "PivotTable report is not using AutoShow for this fie
    End If
End With
```



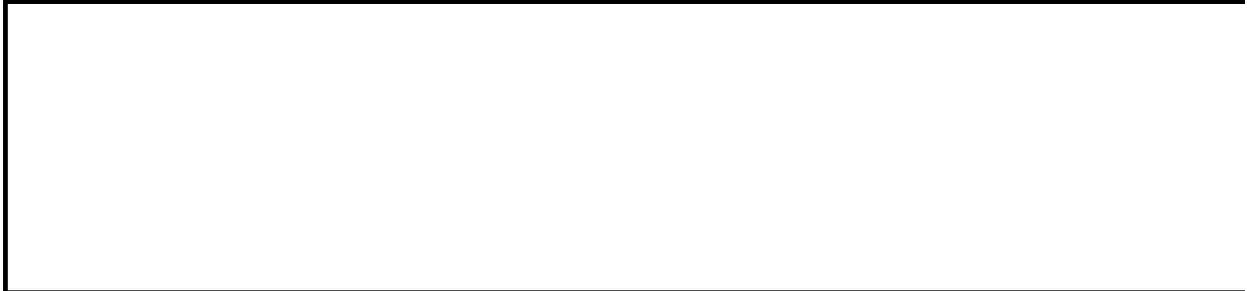
# AutoShowType Property

Returns **xlAutomatic** if **AutoShow** is enabled for the specified PivotTable field; returns **xlManual** if **AutoShow** is disabled. Read-only **Long**.

## Example

This example displays a message box showing the **AutoShow** parameters for the Salesman field.

```
With Worksheets(1).PivotTables(1).PivotFields("salesman")
  If .AutoShowType = xlAutomatic Then
    r = .AutoShowRange
    If r = xlTop Then
      rn = "top"
    Else
      rn = "bottom"
    End If
    MsgBox "PivotTable report is showing " & rn & " " & _
      .AutoShowCount & " items in " & .Name & _
      " field by " & .AutoShowField
  Else
    MsgBox "PivotTable report is not using AutoShow for this fie
  End If
End With
```



# AutoSize Property

**True** if the size of the specified object is changed automatically to fit text within its boundaries. Read/write **Boolean**.

## Example

This example adjusts the size of the text frame on shape one to fit its text.

```
worksheets(1).Shapes(1).TextFrame.AutoSize = True
```



# AutoSortField Property

Returns the name of the data field used to sort the specified PivotTable field automatically. Read-only **String**.

## Example

This example displays a message box showing the AutoSort parameters for the Product field.

```
With Worksheets(1).PivotTables(1).PivotFields("product")
    Select Case .AutoSortOrder
        Case xlManual
            aso = "manual"
        Case xlAscending
            aso = "ascending"
        Case xlDescending
            aso = "descending"
    End Select
    MsgBox " sorted in " & aso & _
        " order by " & .AutoSortField
End With
```



[Show All](#)

# AutoSortOrder Property

Returns the order used to sort the specified PivotTable field automatically. Can be one of the following [XISortOrder](#) constants. Read-only **Long**.

XISortOrder can be one of these XISortOrder constants.

**XIAscending**

**XIDescending**

**XIManual**. If automatic sorting is disabled.

*expression*.**AutoSortOrder**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example displays a message box showing the AutoSort parameters for the Product field.

```
With Worksheets(1).PivotTables(1).PivotFields("product")
    Select Case .AutoSortOrder
        Case xlManual
            aso = "manual"
        Case xlAscending
            aso = "ascending"
        Case xlDescending
            aso = "descending"
    End Select
    MsgBox " sorted in " & aso & _
        " order by " & .AutoSortField
End With
```



# AutoText Property

**True** if the object automatically generates appropriate text based on context.  
Read/write **Boolean**.

## Example

This example sets the data labels for series one in Chart1 to automatically generate appropriate text.

```
Charts("Chart1").SeriesCollection(1).DataLabels.AutoText = True
```



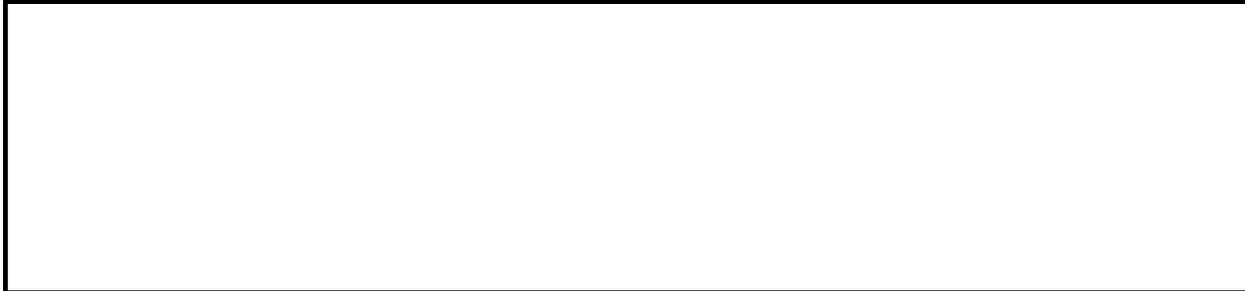
# AutoUpdate Property

**True** if the OLE object is updated automatically when the source changes. Valid only if the object is linked (its **OLEType** property must be **xIOLELink**). Read-only **Boolean**.

## Example

This example displays the status of automatic updating for all OLE objects on Sheet1.

```
Worksheets("Sheet1").Activate
Range("A1").Value = "Name"
Range("B1").Value = "Link Status"
Range("C1").Value = "AutoUpdate Status"
i = 2
For Each obj In ActiveSheet.OLEObjects
    Cells(i, 1) = obj.Name
    If obj.OLEType = xlOLELink Then
        Cells(i, 2) = "Linked"
        Cells(i, 3) = obj.AutoUpdate
    Else
        Cells(i, 2) = "Embedded"
    End If
    i = i + 1
Next
```



# AutoUpdateFrequency Property

Returns or sets the number of minutes between automatic updates to the shared workbook. Read/write **Long**.

## Remarks

The **AutoUpdateFrequency** property must be set to a value from 5 to 1440 for this property to take effect.

## Example

This example causes the shared workbook to be automatically updated every five minutes.

```
ActiveWorkbook.AutoUpdateFrequency = 5
```



# AutoUpdateSaveChanges Property

**True** if current changes to the shared workbook are posted to other users whenever the workbook is automatically updated. **False** if changes aren't posted (this workbook is still synchronized with changes made by other users). The default value is **True**. Read/write **Boolean**.

## Remarks

The **AutoUpdateFrequency** property must be set to a value from 5 to 1440 for this property to take effect.

## Example

This example causes changes to the shared workbook to be posted to other users whenever the workbook is automatically updated.

```
ActiveWorkbook.AutoUpdateSaveChanges = True
```



# AxisBetweenCategories Property

**True** if the value axis crosses the category axis between categories. Read/write **Boolean**.

## Remarks

This property applies only to category axes, and it doesn't apply to 3-D charts.

## Example

This example causes the value axis in Chart1 to cross the category axis between categories.

```
Charts("Chart1").Axes(xlCategory).AxisBetweenCategories = True
```



[Show All](#)

# AxisGroup Property

 [AxisGroup property as it applies to the \*\*ChartGroup\*\* and \*\*Series\*\* objects.](#)

Returns the group for the specified chart group or series. Read/write **XlAxisGroup**.

XlAxisGroup can be one of these XlAxisGroup constants.

**xlPrimary**

**xlSecondary**

*expression*.**AxisGroup**

*expression* Required. An expression that returns one of the above objects.

 [AxisGroup property as it applies to the \*\*Axis\*\* object.](#)

Returns the group for the specified axis. Read-only **XlAxisGroup**.

XlAxisGroup can be one of these XlAxisGroup constants.

**xlPrimary**

**xlSecondary**

*expression*.**AxisGroup**

*expression* Required. An expression that returns one of the above objects.

## Remarks

For 3-D charts, only **xlPrimary** is valid.

## Example

This example deletes the value axis in Chart1 if the axis is in the secondary group.

```
With Charts("Chart1").Axes(xlValue)  
    If .AxisGroup = xlSecondary Then .Delete  
End With
```



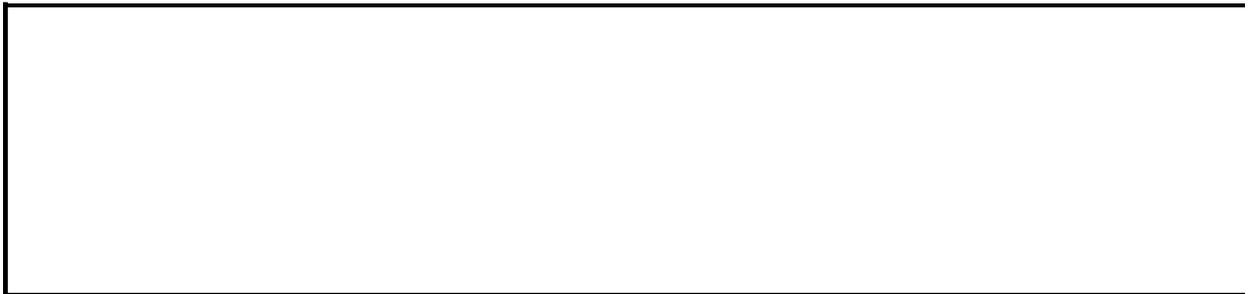
# AxisTitle Property

Returns an [AxisTitle](#) object that represents the title of the specified axis. Read-only.

## Example

This example adds an axis label to the category axis in Chart1.

```
With Charts("Chart1").Axes(xlCategory)
    .HasTitle = True
    .AxisTitle.Text = "July Sales"
End With
```



[Show All](#)

# BackColor Property

 [BackColor property as it applies to the \*\*ChartFillFormat\*\* object.](#)

Returns a [ChartColorFormat](#) object that represents the specified fill background color. Read-only **ChartColorFormat** object.

*expression*.**BackColor**

*expression* Required. An expression that returns one of the above objects.

 [BackColor property as it applies to the \*\*FillFormat\*\* and \*\*LineFormat\*\* objects.](#)

Returns a [ColorFormat](#) object that represents the specified fill background color. Read/write **ColorFormat** object.

*expression*.**BackColor**

*expression* Required. An expression that returns one of the above objects.

## Example

This example sets the foreground color, background color, and gradient for the chart area fill on chart one.

```
With Charts(1).ChartArea.Fill  
    .Visible = True  
    .ForeColor.SchemeColor = 15  
    .BackColor.SchemeColor = 17  
    .TwoColorGradient msoGradientHorizontal, 1  
End With
```



[Show All](#)

# Background Property

 [Background property as it applies to the \*\*CanvasShapes\*\* object.](#)

Returns a **Shape** object that represents the background image for the specified document. Read-only.

*expression*.**Background**

*expression* Required. An expression that returns a **CanvasShapes** object.

 [Background property as it applies to the \*\*Font\*\* object.](#)

Returns or sets the text background type. This property is used for text on charts. Read/write **Variant**.

*expression*.**Background**

*expression* Required. An expression that returns a **Font** object.

## Remarks

The following constants can be used with the **Background** property as it applies to the **Font** object: **xlBackgroundAutomatic**, **xlBackgroundOpaque**, **xlBackgroundTransparent**.

## Example

This example adds a chart title to embedded chart one on the first worksheet and then sets the font size and background type for the title. This example assumes a chart exists on the first worksheet.

```
Sub UseBackground()  
  
    With Worksheets(1).ChartObjects(1).Chart  
        .HasTitle = True  
        .ChartTitle.Text = "Rainfall Totals by Month"  
        With .ChartTitle.Font  
            .Size = 10  
            .Background = xlBackgroundTransparent  
        End With  
    End With  
End Sub
```



# BackgroundChecking Property

Alerts the user for all cells that violate enabled error-checking rules. When this property is set to **True** (default), the **AutoCorrect Options** button appears next to all cells that violate enabled errors. **False** disables background checking for errors. Read/write **Boolean**.

*expression*.**BackgroundChecking**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Refer to the [ErrorCheckingOptions](#) object to view a list of its members that can be enabled.

## Example

In the following example, when the user selects cell A1 (which contains a formula referring to empty cells), the **AutoCorrect Options** button appears.

```
Sub CheckBackground()  
    ' Simulate an error by referring to empty cells.  
    Application.ErrorCheckingOptions.BackgroundChecking = True  
    Range("A1").Select  
    ActiveCell.Formula = "=A2/A3"  
End Sub
```



[Show All](#)

# BackgroundQuery Property

**True** if queries for the PivotTable report or query table are performed asynchronously (in the background). Read/write **Boolean**.

## Remarks

For [OLAP](#) data sources, this property is read-only and always returns **False**.

## Example

This example causes queries for the first PivotTable report on worksheet one to be performed in the background.

```
Worksheets(1).PivotTables("Pivot1") _  
    .PivotCache.BackgroundQuery = True
```



# Backward Property

Returns or sets the number of periods (or units on a scatter chart) that the trendline extends backward. Read/write **Long**

## Example

This example sets the number of units that the trendline on Chart1 extends forward and backward. The example should be run on a 2-D column chart that contains a single series with a trendline.

```
With Charts("Chart1").SeriesCollection(1).Trendlines(1)  
    .Forward = 5  
    .Backward = .5  
End With
```



# Bar3DGroup Property

Returns a [ChartGroup](#) object that represents the bar chart group on a 3-D chart.  
Read-only.

## Example

This example sets the space between bar clusters in the 3-D bar chart group to be 50 percent of the bar width.

```
Charts(1).BarGroup3DGroup.GapWidth = 50
```



[Show All](#)

# BarShape Property

Returns or sets the shape used with the 3-D bar or column chart. Read/write [XlBarShape](#).

XlBarShape can be one of these XlBarShape constants.

**xlBox**

**xlConeToPoint**

**xlPyramidToMax**

**xlConeToMax**

**xlCylinder**

**xlPyramidToPoint**

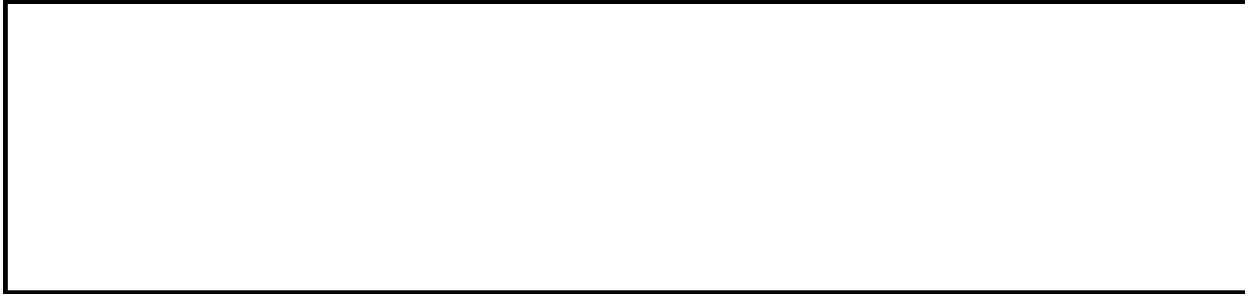
*expression*.**BarShape**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example sets the shape used with series one on chart one.

```
Charts(1).SeriesCollection(1).BarShape = xlConeToPoint
```



[Show All](#)

# BaseField Property

Returns or sets the base field for a custom calculation. This property is valid only for data fields. Read/write **Variant**.

## Remarks

This property is not available for [OLAP](#) data sources.

## Example

This example sets the data field in the PivotTable report on Sheet1 to calculate the difference from the base field, sets the base field to the field named "ORDER\_DATE," and then sets the base item to the item named "5/16/89."

```
With Worksheets("Sheet1").Range("A3").PivotField
    .Calculation = xlDifferenceFrom
    .BaseField = "ORDER_DATE"
    .BaseItem = "5/16/89"
End With
```



[Show All](#)

# BaseItem Property

Returns or sets the item in the base field for a custom calculation. Valid only for data fields. Read/write **Variant**.

## Remarks

This property is not available for [OLAP](#) data sources.

## Example

This example sets the data field in the PivotTable report on Sheet1 to calculate the difference from the base field, sets the base field to the field named "ORDER\_DATE," and then sets the base item to the item named "5/16/89."

```
With Worksheets("Sheet1").Range("A3").PivotField
    .Calculation = xlDifferenceFrom
    .BaseField = "ORDER_DATE"
    .BaseItem = "5/16/89"
End With
```



[Show All](#)

# BaseUnit Property

Returns or sets the base unit for the specified category axis. Read/write [XLTimeUnit](#).

XLTimeUnit can be one of these XLTimeUnit constants.

**xlMonths**

**xlDays**

**xlYears**

*expression*.**BaseUnit**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Setting this property has no visible effect if the **CategoryType** property for the specified axis is set to **xlCategoryScale**. The set value is retained, however, and takes effect when the **CategoryType** property is set to **xlTimeScale**.

You cannot set this property for a value axis.

## Example

This example sets the category axis in embedded chart one on worksheet one to use a time scale, with months as the base unit.

```
With Worksheets(1).ChartObjects(1).Chart
    With .Axes(xlCategory)
        .CategoryType = xlTimeScale
        .BaseUnit = xlMonths
    End With
End With
```



# BaseUnitIsAuto Property

**True** if Microsoft Excel chooses appropriate base units for the specified category axis. The default value is **True**. Read/write **Boolean**.

## Remarks

You cannot set this property for a value axis.

## Example

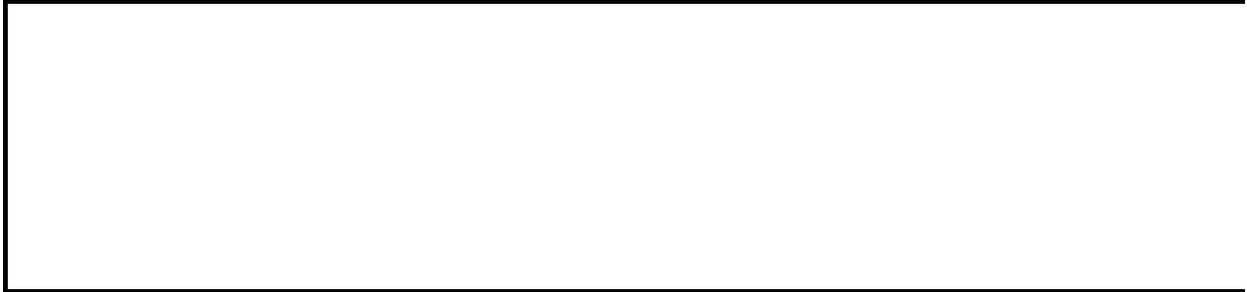
This example sets the category axis in embedded chart one on worksheet one to use a time scale with automatic base units.

```
With Worksheets(1).ChartObjects(1).Chart
    With .Axes(xlCategory)
        .CategoryType = xlTimeScale
        .BaseUnitIsAuto = True
    End With
End With
```



# **BCCRecipients Property**

You have requested Help for a Visual Basic keyword used only on the Macintosh. For information about this keyword, consult the language reference Help included with Microsoft Office Macintosh Edition.



[Show All](#)

# BeginArrowheadLength Property

Returns or sets the length of the arrowhead at the beginning of the specified line.  
Read/write [MsoArrowheadLength](#).

MsoArrowheadLength can be one of these MsoArrowheadLength constants.

**msoArrowheadLengthMixed**

**msoArrowheadShort**

**msoArrowheadLengthMedium**

**msoArrowheadLong**

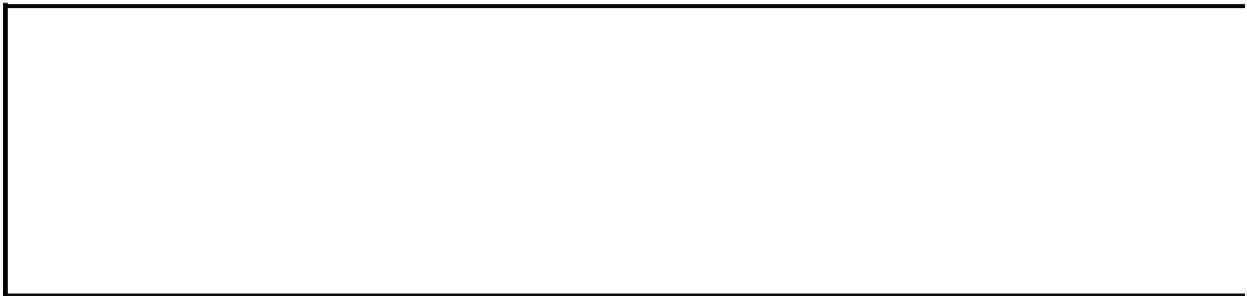
*expression*.**BeginArrowheadLength**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example adds a line to myDocument. There's a short, narrow oval on the line's starting point and a long, wide triangle on its end point.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddLine(100, 100, 200, 300).Line
    .BeginArrowheadLength = msoArrowheadShort
    .BeginArrowheadStyle = msoArrowheadOval
    .BeginArrowheadWidth = msoArrowheadNarrow
    .EndArrowheadLength = msoArrowheadLong
    .EndArrowheadStyle = msoArrowheadTriangle
    .EndArrowheadWidth = msoArrowheadWide
End With
```



[Show All](#)

# BeginArrowheadStyle Property

Returns or sets the style of the arrowhead at the beginning of the specified line.  
Read/write [MsoArrowheadStyle](#).

MsoArrowheadStyle can be one of these MsoArrowheadStyle constants.

**msoArrowheadNone**

**msoArrowheadOval**

**msoArrowheadStyleMixed**

**msoArrowheadDiamond**

**msoArrowheadOpen**

**msoArrowheadStealth**

**msoArrowheadTriangle**

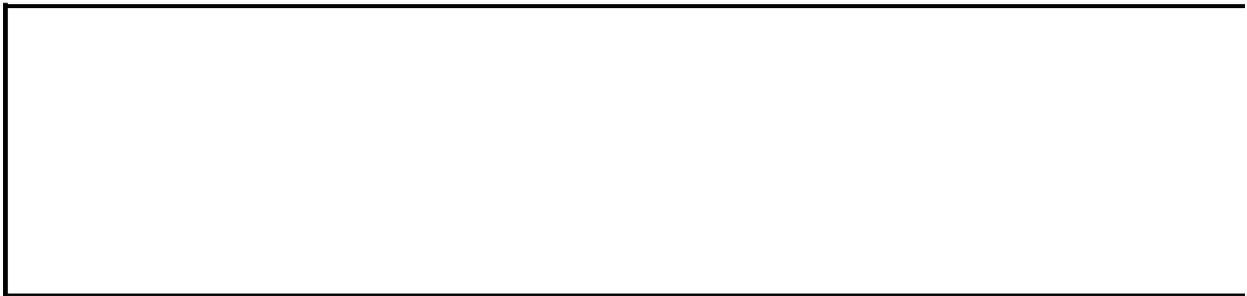
*expression*.**BeginArrowheadStyle**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example adds a line to myDocument. There's a short, narrow oval on the line's starting point and a long, wide triangle on its end point.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddLine(100, 100, 200, 300).Line
    .BeginArrowheadLength = msoArrowheadShort
    .BeginArrowheadStyle = msoArrowheadOval
    .BeginArrowheadWidth = msoArrowheadNarrow
    .EndArrowheadLength = msoArrowheadLong
    .EndArrowheadStyle = msoArrowheadTriangle
    .EndArrowheadWidth = msoArrowheadWide
End With
```



[Show All](#)

# BeginArrowheadWidth Property

Returns or sets the width of the arrowhead at the beginning of the specified line.  
Read/write [MsoArrowheadWidth](#).

MsoArrowheadWidth can be one of these MsoArrowheadWidth constants.

**msoArrowheadNarrow**

**msoArrowheadWidthMedium**

**msoArrowheadWide**

**msoArrowheadWidthMixed**

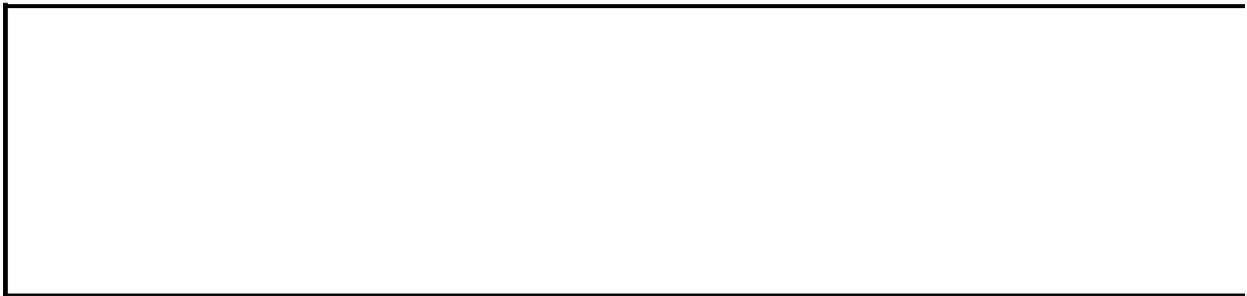
*expression*.**BeginArrowheadWidth**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example adds a line to myDocument. There's a short, narrow oval on the line's starting point and a long, wide triangle on its end point.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddLine(100, 100, 200, 300).Line
    .BeginArrowheadLength = msoArrowheadShort
    .BeginArrowheadStyle = msoArrowheadOval
    .BeginArrowheadWidth = msoArrowheadNarrow
    .EndArrowheadLength = msoArrowheadLong
    .EndArrowheadStyle = msoArrowheadTriangle
    .EndArrowheadWidth = msoArrowheadWide
End With
```



[Show All](#)

# BeginConnected Property

**True** if the beginning of the specified connector is connected to a shape. Read-only [MsoTriState](#).

MsoTriState can be one of these MsoTriState constants.

**msoCTrue**

**msoFalse**

**msoTriStateMixed**

**msoTriStateToggle**

**msoTrue** The beginning of the specified connector is connected to a shape.

*expression*.**BeginConnected**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

If shape three on myDocument is a connector whose beginning is connected to a shape, this example stores the connection site number in the variable oldBeginConnSite, stores a reference to the connected shape in the object variable oldBeginConnShape, and then disconnects the beginning of the connector from the shape.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(3)
    If .Connector Then
        With .ConnectorFormat
            If .BeginConnected Then
                oldBeginConnSite = .BeginConnectionSite
                Set oldBeginConnShape = .BeginConnectedShape
                .BeginDisconnect
            End If
        End With
    End If
End With
```



# BeginConnectedShape Property

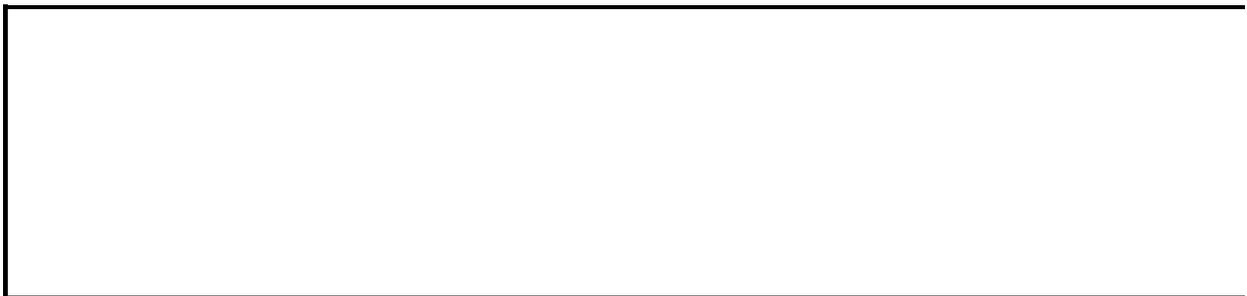
Returns a [Shape](#) object that represents the shape that the beginning of the specified connector is attached to. Read-only.

**Note** If the beginning of the specified connector isn't attached to a shape, this property generates an error.

## Example

This example assumes that myDocument already contains two shapes attached by a connector named "Conn1To2." The code adds a rectangle and a connector to myDocument. The beginning of the new connector will be attached to the same connection site as the beginning of the connector named "Conn1To2," and the end of the new connector will be attached to connection site one on the new rectangle.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes
    Set r3 = .AddShape(msoShapeRectangle, 450, 190, 200, 100)
    .AddConnector(msoConnectorCurve, 0, 0, 10, 10).Name = _
        "Conn1To3"
    With .Item("Conn1To2").ConnectorFormat
        beginConnSite1 = .BeginConnectionSite
        Set beginConnShape1 = .BeginConnectedShape
    End With
    With .Item("Conn1To3").ConnectorFormat
        .BeginConnect beginConnShape1, beginConnSite1
        .EndConnect r3, 1
    End With
End With
```



# BeginConnectionSite Property

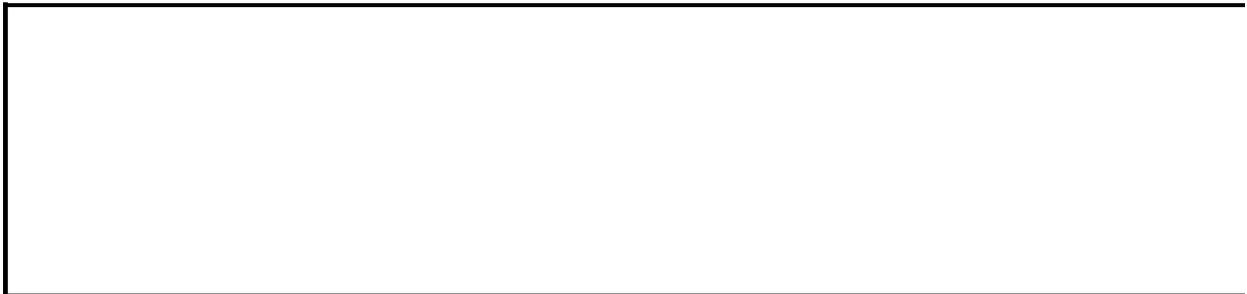
Returns an integer that specifies the connection site that the beginning of a connector is connected to. Read-only **Long**.

**Note** If the beginning of the specified connector isn't attached to a shape, this property generates an error.

## Example

This example assumes that myDocument already contains two shapes attached by a connector named "Conn1To2." The code adds a rectangle and a connector to myDocument. The beginning of the new connector will be attached to the same connection site as the beginning of the connector named "Conn1To2," and the end of the new connector will be attached to connection site one on the new rectangle.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes
    Set r3 = .AddShape(msoShapeRectangle, 450, 190, 200, 100)
    .AddConnector(msoConnectorCurve, 0, 0, 10, 10).Name = _
        "Conn1To3"
    With .Item("Conn1To2").ConnectorFormat
        beginConnSite1 = .BeginConnectionSite
        Set beginConnShape1 = .BeginConnectedShape
    End With
    With .Item("Conn1To3").ConnectorFormat
        .BeginConnect beginConnShape1, beginConnSite1
        .EndConnect r3, 1
    End With
End With
```



# **BlackAndWhite Property**

**True** if elements of the document will be printed in black and white. Read/write **Boolean**.

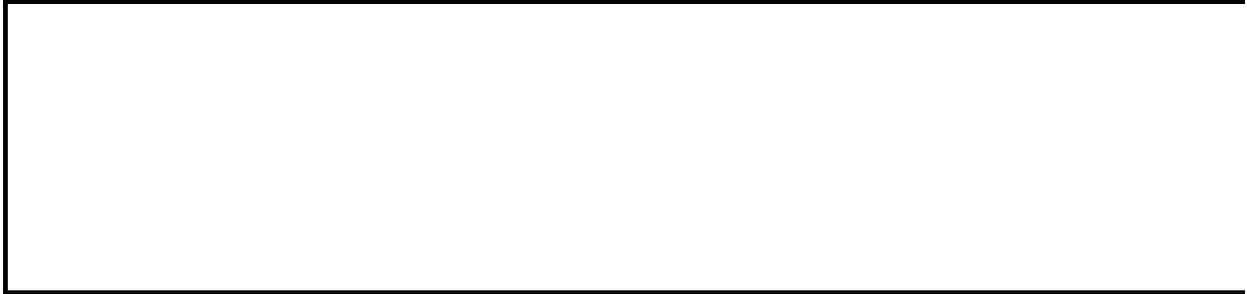
## **Remarks**

This property applies only to worksheet pages.

## Example

This example causes Sheet1 to be printed in black and white.

```
Worksheets("Sheet1").PageSetup.BlackAndWhite = True
```



[Show All](#)

# BlackWhiteMode Property

Returns or sets a value that indicates how the specified shape appears when the presentation is viewed in black-and-white mode. Read/write [MsoBlackWhiteMode](#).

MsoBlackWhiteMode can be one of these MsoBlackWhiteMode constants.

**msoBlackWhiteAutomatic**

**msoBlackWhiteBlack**

**msoBlackWhiteBlackTextAndLine**

**msoBlackWhiteDontShow**

**msoBlackWhiteGrayOutline**

**msoBlackWhiteGrayScale**

**msoBlackWhiteHighContrast**

**msoBlackWhiteInverseGrayScale**

**msoBlackWhiteLightGrayScale**

**msoBlackWhiteMixed**

**msoBlackWhiteWhite**

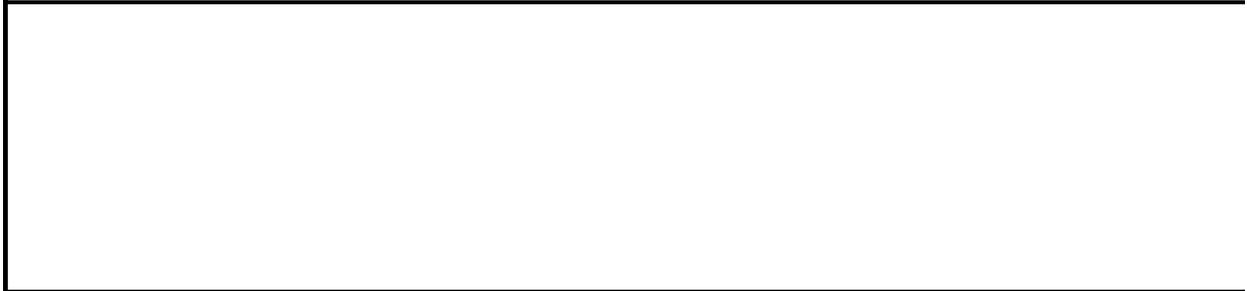
*expression*.**BlackWhiteMode**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example sets shape one on wksOne to appear in black-and-white mode. When you view the presentation in black-and-white mode, shape one will appear black regardless of what color it is in color mode.

```
Sub UseBlackWhiteMode()  
  
    Dim wksOne As Worksheet  
    Set wksOne = Application.Worksheets(1)  
    wksOne.Shapes(1).BlackWhiteMode = msoBlackWhiteGrayOutline  
  
End Sub
```



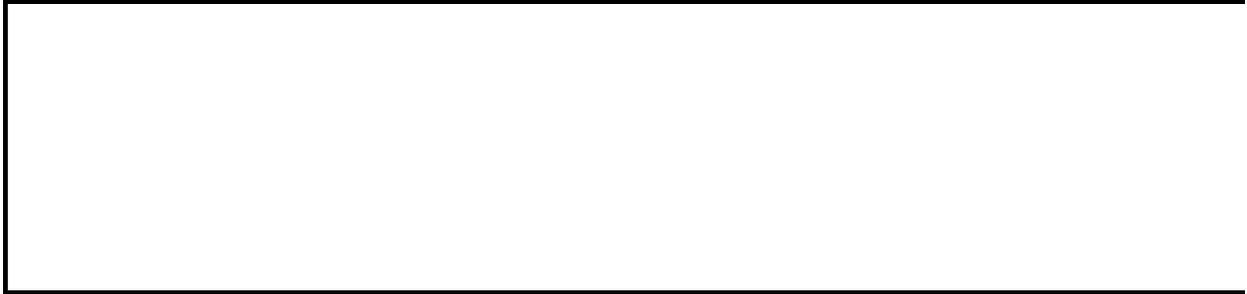
# **Bold Property**

**True** if the font is bold. Read/write **Variant**.

## Example

This example sets the font to bold for the range A1:A5 on Sheet1.

```
Worksheets("Sheet1").Range("A1:A5").Font.Bold = True
```



[Show All](#)

# Border Property

 [Border property as it applies to the \*\*CalloutFormat\*\* object.](#)

Represents the visibility options for the border of the object. Read/write **MsoTriState**.

MsoTriState can be one of these MsoTriState constants.

**msoCTrue** Does not apply to this object.

**msoFalse** Sets the border invisible.

**msoTriStateMixed** Does not apply to this object.

**msoTriStateToggle** Allows the user to switch the border from visible to invisible and vice versa.

**msoTrue** *default*. Sets the border visible.

*expression*.**Border**

*expression* Required. An expression that returns a **CallFormat** object.

 [Border property as it applies to all other objects in the Applies To list.](#)

Returns a **Border** object that represents the border of the object.

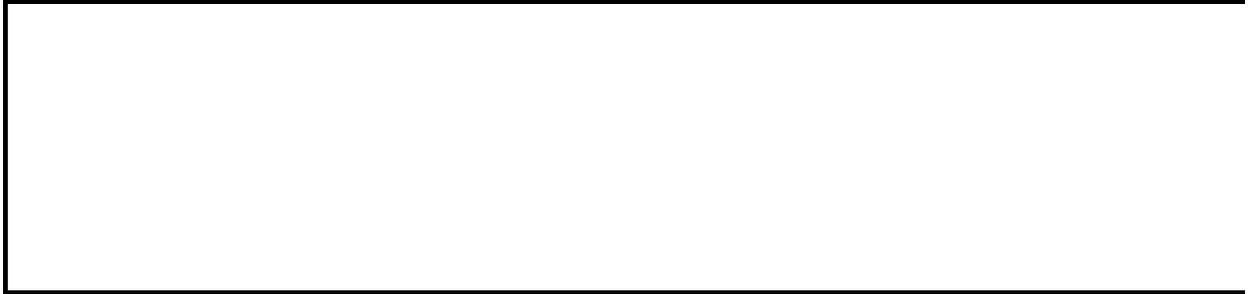
*expression*.**Border**

*expression* Required. An expression that returns all other objects in the Applies To list.

## Example

This example sets the color of the chart area border of Chart1 to red.

```
Charts("Chart1").ChartArea.Border.ColorIndex = 3
```



[Show All](#)

# Borders Property

 [Borders property as it applies to the \*\*CellFormat\*\* object.](#)

Allows the user to set or return the search criteria based on the cell's border format.

*expression*.**Borders**

*expression* Required. An expression that returns a **CellFormat** object.

 [Borders property as it applies to the \*\*FormatCondition, Range, and Style\*\* objects.](#)

Returns a **Borders** collection that represents the borders of a style or a range of cells (including a range defined as part of a conditional format).

*expression*.**Borders**

*expression* Required. An expression that returns one of the above objects.

## Remarks

For information about returning a single member of a collection, see [Returning an Object from a Collection](#).

## Example

 [As it applies to the \*\*CellFormat\*\* object.](#)

This example sets the search criteria to identify the borders of cells that have a continuous and thick style bottom-edge, creates a cell with this condition, finds this cell, and notifies the user.

**Note** The default color of the border is used in this example, therefore the color index is not changed.

```
Sub SearchCellFormat()  
  
    ' Set the search criteria for the border of the cell format.  
    With Application.FindFormat.Borders(xlEdgeBottom)  
        .LineStyle = xlContinuous  
        .Weight = xlThick  
    End With  
  
    ' Create a continuous thick bottom-edge border for cell A5.  
    Range("A5").Select  
    With Selection.Borders(xlEdgeBottom)  
        .LineStyle = xlContinuous  
        .Weight = xlThick  
    End With  
    Range("A1").Select  
    MsgBox "Cell A5 has a continuous thick bottom-edge border"  
  
    ' Find the cells based on the search criteria.  
    Cells.Find(What:="", After:=ActiveCell, LookIn:=xlFormulas, Look  
        xlPart, SearchOrder:=xlByRows, SearchDirection:=xlNext, Matc  
        , SearchFormat:=True).Activate  
    MsgBox "Microsoft Excel has found this cell matching the search"  
  
End Sub
```

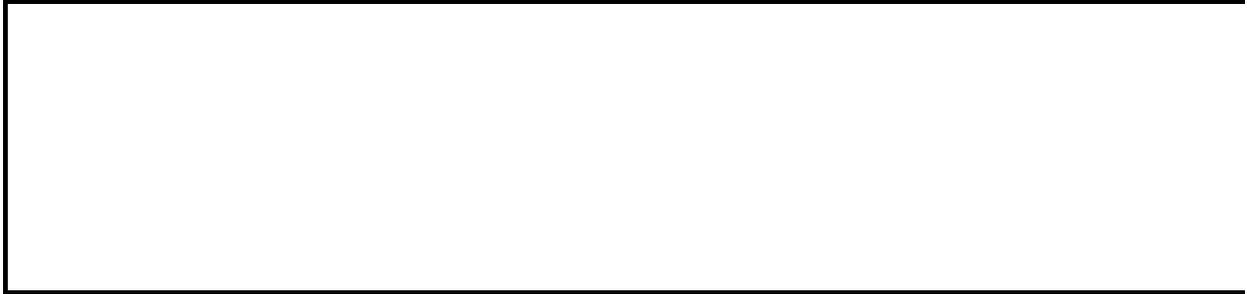
 [As it applies to the \*\*FormatCondition\*\*, \*\*Range\*\*, and \*\*Style\*\* objects.](#)

This example sets the color of the bottom border of cell B2 on Sheet1 to a thin red border.

```
Sub SetRangeBorder()
```

```
With Worksheets("Sheet1").Range("B2").Borders(xlEdgeBottom)  
    .LineStyle = xlContinuous  
    .Weight = xlThin  
    .ColorIndex = 3  
End With
```

End Sub



[Show All](#)

# BottomMargin Property

Returns or sets the size of the bottom margin, in [points](#). Read/write **Double**.

## Remarks

Margins are set or returned in points. Use either the **InchesToPoints** method or the **CentimetersToPoints** method to do the conversion.

## Example

These two examples set the bottom margin of Sheet1 to 0.5 inch (36 points).

```
Worksheets("Sheet1").PageSetup.BottomMargin = _  
    Application.InchesToPoints(0.5)
```

```
Worksheets("Sheet1").PageSetup.BottomMargin = 36
```

This example displays the current setting for the bottom margin on Sheet1.

```
marginInches = Worksheets("Sheet1").PageSetup.BottomMargin / _  
    Application.InchesToPoints(1)  
MsgBox "The current bottom margin is " & marginInches & " inches"
```



# BottomRightCell Property

Returns a [Range](#) object that represents the cell that lies under the lower-right corner of the object. Read-only.

## Example

This example displays the address of the cell beneath the lower-right corner of embedded chart one on Sheet1.

```
MsgBox "The bottom right corner is over cell " & _  
    Worksheets("Sheet1").ChartObjects(1).BottomRightCell.Address
```



# Brightness Property

Returns or sets the brightness of the specified picture or OLE object. The value for this property must be a number from 0.0 (dimkest) to 1.0 (brightest).

Read/write **Single**.

*expression*.**Brightness**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example sets the brightness for shape one on myDocument. Shape one must be either a picture or an OLE object.

```
Set myDocument = Worksheets(1)  
myDocument.Shapes(1).PictureFormat.Brightness = 0.3
```



# BubbleScale Property

Returns or sets the scale factor for bubbles in the specified chart group. Can be an integer value from 0 (zero) to 300, corresponding to a percentage of the default size. Applies only to bubble charts. Read/write **Long**.

## Example

This example sets the bubble size in chart group one to 200% of the default size.

```
With Worksheets(1).ChartObjects(1).Chart  
    .ChartGroups(1).BubbleScale = 200  
End With
```



# BubbleSizes Property

Returns or sets a string that refers to the worksheet cells containing the size data for the bubble chart. When you return the cell reference, it will return a string describing the cells in A1-style notation. To set the size data for the bubble chart, you must use R1-style notation. Applies only to bubble charts. Read/write **Variant**.

## Example

This example displays the cell reference for the cells that contain the bubble chart size data.

```
MsgBox Worksheets(1).ChartObjects(1).Chart _  
    .SeriesCollection(1).BubbleSizes
```

This example shows how to set this property using R1-style notation.

```
Worksheets(1).ChartObjects(1).Chart _  
    .SeriesCollection(1).BubbleSizes = "=Sheet1!r1c5:r5c5"
```



# Build Property

Returns the Microsoft Excel build number. Read-only **Long**.

## Remarks

It's usually safer to test the **Version** property, unless you're sure you need to know the build number.

## Example

This example tests the **Build** property.

```
If Application.Build > 2500 Then  
    ' build-dependent code here  
End If
```



# BuiltIn Property

**True** if the style is a built-in style. Read-only **Boolean**.

## Example

This example creates a list on worksheet one that contains the names and built-in status of all the styles in the active workbook.

```
r = 0
Worksheets(1).Activate
For Each s In ActiveWorkbook.Styles
    r = r + 1
    Cells(r, 1).Value = s.Name
    Cells(r, 2).Value = s.BuiltIn
Next
```



# BuiltinDocumentProperties Property

Returns a [DocumentProperties](#) collection that represents all the built-in document properties for the specified workbook. Read-only.

## Remarks

This property returns the entire collection of built-in document properties. Use the **Item** method to return a single member of the collection (a **DocumentProperty** object) by specifying either the name of the property or the collection index (as a number).

You can refer to document properties either by index value or by name. The following list shows the available built-in document property names:

Title	Creation Date	Company
Subject	Last Save Time	Number of Bytes
Author	Total Editing Time	Number of Lines
Keywords	Number of Pages	Number of Paragraphs
Comments	Number of Words	Number of Slides
Template	Number of Characters	Number of Notes
Last Author	Security	Number of Hidden Slides
Revision Number	Category	Number of Multimedia Clips
Application Name	Format	Hyperlink Base
Last Print Date	Manager	Number of Characters (with spaces)

Container applications aren't required to define values for every built-in document property. If Microsoft Excel doesn't define a value for one of the built-in document properties, reading the **Value** property for that document property causes an error.

Because the **Item** method is the default method for the **DocumentProperties** collection, the following statements are identical:

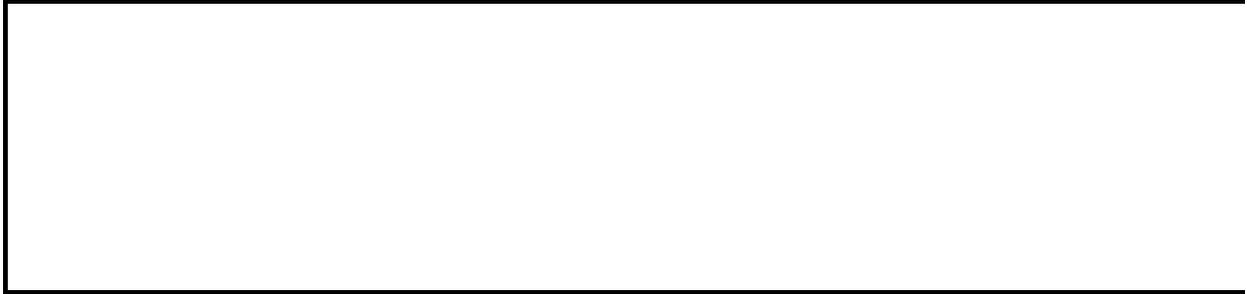
```
BuiltinDocumentProperties.Item(1)  
BuiltinDocumentProperties(1)
```

Use the [CustomDocumentProperties](#) property to return the collection of custom document properties.

## Example

This example displays the names of the built-in document properties as a list on worksheet one.

```
rw = 1
Worksheets(1).Activate
For Each p In ActiveWorkbook.BuiltinDocumentProperties
    Cells(rw, 1).Value = p.Name
    rw = rw + 1
Next
```



# CacheIndex Property

Returns or sets the index number of the PivotTable cache. Read/write **Long**.

## Remarks

If you set the **CacheIndex** property so that one PivotTable report uses the cache for a second PivotTable report, the first report's fields must be a valid subset of the fields in the second report.

## Example

This example sets the cache for the PivotTable report named "Pivot1" to the cache of the PivotTable report named "Pivot2."

```
Worksheets(1).PivotTables("Pivot1").CacheIndex = _  
    Worksheets(1).PivotTables("Pivot2").CacheIndex
```



# CalculateBeforeSave Property

**True** if workbooks are calculated before they're saved to disk (if the [Calculation](#) property is set to **xlManual**). This property is preserved even if you change the **Calculation** property. Read/write **Boolean**.

## Example

This example sets Microsoft Excel to calculate workbooks before they're saved to disk.

```
Application.Calculation = xlManual  
Application.CalculateBeforeSave = True
```



[Show All](#)

# CalculatedMembers Property

Returns a [CalculatedMembers](#) collection representing all the calculated members and calculated measures for an OLAP PivotTable.

*expression*.**CalculatedMembers**

*expression* Required. An expression that returns a [PivotTable](#) object.

## Remarks

This property is used for [Online Analytical Processing \(OLAP\)](#) sources; a non-OLAP source will return a run-time error.

## Example

This example adds a set to the PivotTable. It assumes a PivotTable exists on the active worksheet that is connected to an OLAP data source which contains a field titled "[Product].[All Products]".

```
Sub UseCalculatedMember()  
  
    Dim pvtTable As PivotTable  
  
    Set pvtTable = ActiveSheet.PivotTables(1)  
  
    ' Add the calculated member.  
    pvtTable.CalculatedMembers.Add Name:="[Beef]", _  
        Formula:="'{[Product].[All Products].Children}'", _  
        Type:=xlCalculatedSet  
  
End Sub
```



[Show All](#)

# Calculation Property

 [Calculation property as it applies to the \*\*Application\*\* object.](#)

Returns or sets the calculation mode. Read/write [XlCalculation](#).

XlCalculation can be one of these XlCalculation constants.

**xlCalculationAutomatic**

**xlCalculationManual**

**xlCalculationSemiautomatic**

*expression*.**Calculation**

*expression* Required. An expression that returns one of the above objects.

 [Calculation property as it applies to the \*\*PivotField\*\* object.](#)

Returns or sets the type of calculation performed by the specified field. This property is valid only for data fields. Read/write [XlPivotFieldCalculation](#).

XlPivotFieldCalculation can be one of these XlPivotFieldCalculation constants.

**xlDifferenceFrom**

**xlIndex**

**xlNoAdditionalCalculation**

**xlPercentDifferenceFrom**

**xlPercentOf**

**xlPercentOfColumn**

**xlPercentOfRow**

**xlPercentOfTotal**

**xlRunningTotal**

*expression*.**Calculation**

*expression* Required. An expression that returns one of the above objects.

## Remarks

For [OLAP](#) data sources, this property can only return or be set to **xlNormal**.

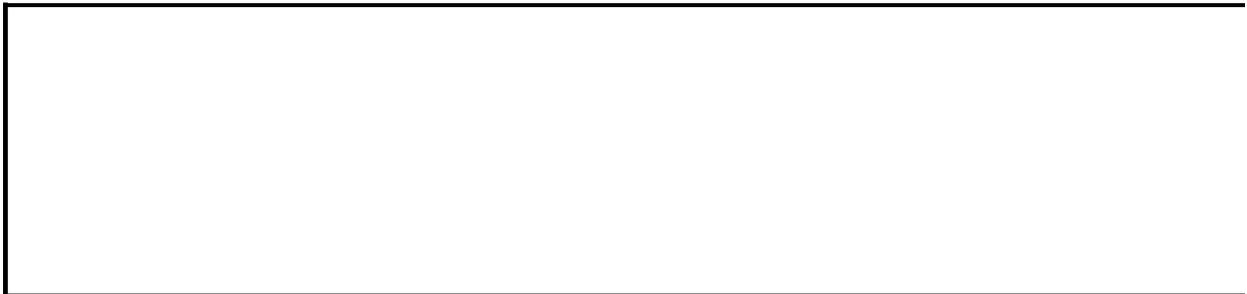
## Example

This example causes Microsoft Excel to calculate workbooks before they are saved to disk.

```
Application.Calculation = xlCalculationManual  
Application.CalculateBeforeSave = True
```

This example sets the data field in the PivotTable report on Sheet1 to calculate the difference from the base field, sets the base field to the field named "ORDER\_DATE," and then sets the base item to the item named "5/16/89."

```
With Worksheets("Sheet1").Range("A3").PivotField  
    .Calculation = xlDifferenceFrom  
    .BaseField = "ORDER_DATE"  
    .BaseItem = "5/16/89"  
End With
```



[Show All](#)

# CalculationInterruptKey Property

Sets or returns an [XlCalculationInterruptKey](#) constant that specifies the key that can interrupt Microsoft Excel when performing calculations. Read/write.

XlCalculationInterruptKey can be one of these XlCalculationInterruptKey constants.

**xlAnyKey**

**xlEscKey**

**xlNoKey**

*expression*.**CalculationInterruptKey**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel determines the setting for the calculation interrupt key and notifies the user.

```
Sub CheckInterruptKey()  
    ' Determine the calculation interrupt key and notify the user.  
    Select Case Application.CalculationInterruptKey  
        Case xlAnyKey  
            MsgBox "The calculation interrupt key is set to any key."  
        Case xlEscKey  
            MsgBox "The calculation interrupt key is set to 'Escape'"  
        Case xlNoKey  
            MsgBox "The calculation interrupt key is set to no key."  
    End Select  
End Sub
```



[Show All](#)

# CalculationState Property

Returns an [XICalculationState](#) constant that indicates the calculation state of the application, for any calculations that are being performed in Microsoft Excel. Read-only.

XICalculationState can be one of these XICalculationState constants.

**xICalculating**

**xIDone**

**xIPending**

*expression*.**CalculationState**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel checks to see if any calculations are being performed. If no calculations are being performed, a message displays the calculation state as "Done". Otherwise, a message displays the calculation state as "Not Done".

```
Sub StillCalculating()  
    If Application.CalculationState = xlDone Then  
        MsgBox "Done"  
    Else  
        MsgBox "Not Done"  
    End If  
End Sub
```



# CalculationVersion Property

Returns a number whose rightmost four digits are the minor calculation engine version number, and whose other digits (on the left) are the major version of Microsoft Excel. For a [Workbook](#) object, this property returns the information about the version of Excel that the workbook was last fully recalculated by. Read-only **Long**.

*expression*.**CalculationVersion**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

If the workbook was saved in an earlier version of Excel and if the workbook hasn't been fully recalculated, then this property returns 0.

## Example

This example compares the version of Microsoft Excel with the version of Excel that the workbook was last calculated in. If the two version numbers are different, the example sets the `blnFullCalc` variable to **True**.

```
If Application.CalculationVersion <> _  
    Workbooks(1).CalculationVersion Then  
    blnFullCalc = True  
Else  
    blnFullCalc = False  
End If
```



# Caller Property

Returns information about how Visual Basic was called (for more information, see the Remarks section).

*expression*.**Caller**(*Index*)

*expression* Required. An expression that returns an **Application** object.

*Index* Optional **Variant**. An index to the array. This argument is used only when the property returns an array (for more information, see the Remarks section).

## Remarks

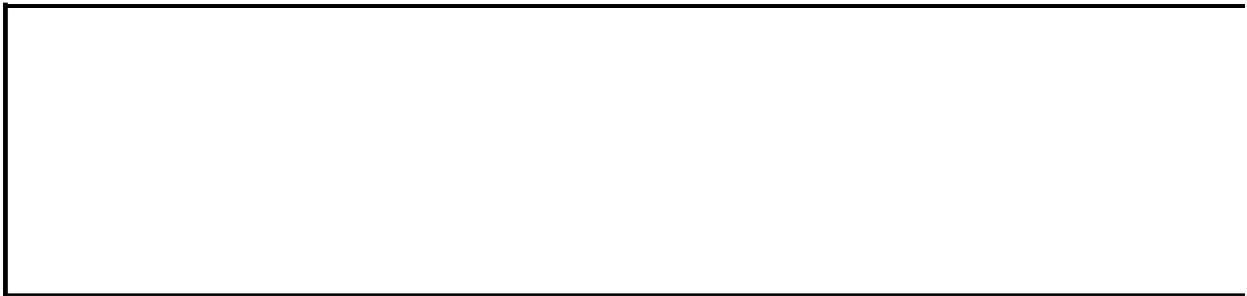
This property returns information about how Visual Basic was called, as shown in the following table.

Caller	Return value
A custom function entered in a single cell	A <b>Range</b> object specifying that cell
A custom function that is part of an array formula in a range of cells	A <b>Range</b> object specifying that range of cells
An Auto_Open, Auto_Close, Auto_Activate, or Auto_Deactivate macro	The name of the document as text
A macro set by either the <b>OnDoubleClick</b> or <b>OnEntry</b> property	The name of the chart object identifier or cell reference (if applicable) to which the macro applies
The <b>Macro</b> dialog box ( <b>Tools</b> menu), or any caller not described above	The #REF! error value

## Example

This example displays information about how Visual Basic was called.

```
Select Case TypeName(Application.Caller)
  Case "Range"
    v = Application.Caller.Address
  Case "String"
    v = Application.Caller
  Case "Error"
    v = "Error"
  Case Else
    v = "unknown"
End Select
MsgBox "caller = " & v
```



# Callout Property

Returns a [CalloutFormat](#) object that contains callout formatting properties for the specified shape. Applies to **Shape** or **ShapeRange** objects that represent line callouts. Read-only.

## Example

This example adds to myDocument an oval and a callout that points to the oval. The callout text won't have a border, but it will have a vertical accent bar that separates the text from the callout line.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes
    .AddShape msoShapeOval, 180, 200, 280, 130
    With .AddCallout(msoCalloutTwo, 420, 170, 170, 40)
        .TextFrame.Characters.Text = "My oval"
        With .Callout
            .Accent = True
            .Border = False
        End With
    End With
End With
```



# CanPlaySounds Property

This property should not be used. Sound notes have been removed from Microsoft Excel.



# CanRecordSounds Property

This property should not be used. Sound notes have been removed from Microsoft Excel.



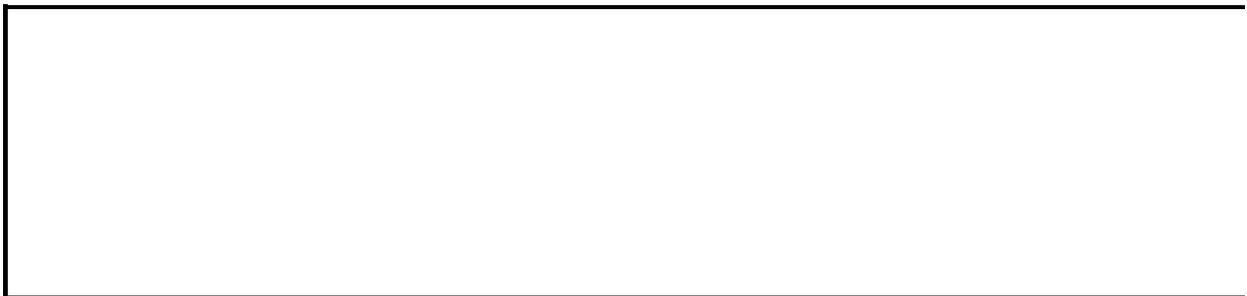
# CapitalizeNamesOfDays Property

**True** if the first letter of day names is capitalized automatically. Read/write **Boolean**.

## Example

This example sets Microsoft Excel to capitalize the first letter of the names of days.

```
With Application.AutoCorrect  
    .CapitalizeNamesOfDays = True  
    .ReplaceText = True  
End With
```



[Show All](#)

# Caption Property

[Caption property as it applies to the \*\*Application\*\* objects.](#)

The name that appears in the title bar of the main Microsoft Excel window. If you don't set a name, or if you set the name to **Empty**, this property returns "Microsoft Excel". Read/write **String**.

*expression*.**Caption**

*expression* Required. An expression that returns one of the above objects.

[Caption property as it applies to the \*\*AxisTitle\*\* objects.](#)

The axis title text. Read/write **String**.

*expression*.**Caption**

*expression* Required. An expression that returns one of the above objects.

[Caption property as it applies to the \*\*Characters\*\* object.](#)

The text of this range of characters. Read-only **String**.

*expression*.**Caption**

*expression* Required. An expression that returns one of the above objects.

[Caption property as it applies to the \*\*ChartTitle\*\* object.](#)

The chart title text. Read-only **String**.

*expression*.**Caption**

*expression* Required. An expression that returns one of the above objects.

[Caption property as it applies to the \*\*DataLabel\*\* object.](#)

The data label text. Read-only **String**.

*expression.Caption*

*expression* Required. An expression that returns one of the above objects.

[Caption property as it applies to the \*\*DisplayUnitLabel\*\* object.](#)

The display unit label text. Read-only **String**.

*expression.Caption*

*expression* Required. An expression that returns one of the above objects.

[Caption property as it applies to the \*\*PivotField\*\* object.](#)

The label text for the pivot field. Read-only **String**.

*expression.Caption*

*expression* Required. An expression that returns one of the above objects.

[Caption property as it applies to the \*\*PivotItem\*\* object.](#)

The label text for the pivot item. Read-only **String**.

*expression.Caption*

*expression* Required. An expression that returns one of the above objects.

[Caption property as it applies to the \*\*CubeField\*\* object.](#)

The label text for the cube field. Read-only **String**.

*expression.Caption*

*expression* Required. An expression that returns one of the above objects.

[Caption property as it applies to the \*\*Window\*\* object.](#)

The name that appears in the title bar of the document window. When you set the name, you can use that name as the index to the **Windows** property (see the second example). Read/write **Variant**.

*expression*.**Caption**

*expression* Required. An expression that returns one of the above objects.

## Remarks

The following table shows example values of the **Caption** property and related properties, given an [OLAP](#) data source with the unique name "[Europe].[France].[Paris]" and a non-OLAP data source with the item name "Paris".

<b>Property</b>	<b>Value (OLAP data source)</b>	<b>Value (non-OLAP data source)</b>
<b>Caption</b>	Paris	Paris
<b>Name</b>	[Europe].[France].[Paris] (read-only)	Paris
<b>SourceName</b>	[Europe].[France].[Paris] (read-only)	(Same as the SQL property value; read-only)
<b>Value</b>	[Europe].[France].[Paris] (read-only)	Paris

When specifying an index into the [PivotItems](#) collection, you can use the syntax shown in the following table.

<b>Syntax (OLAP data source)</b>	<b>Syntax (non-OLAP data source)</b>
expression.PivotItems("[Europe].[France].[Paris]")	expression.PivotItems("Paris")

When using the [Item](#) property to reference a specific member of a collection, you can use the text index names shown in the following table.

<b>Name (OLAP data source)</b>	<b>Name (non-OLAP data source)</b>
[Europe].[France].[Paris]	Paris

## Example

This example sets the name that appears in the title bar of the main Microsoft Excel window to be a custom name.

```
Application.Caption = "Blue Sky Airlines Reservation System"
```

This example sets the name of the first window in the active workbook to be "Consolidated Balance Sheet." This name is then used as the index to the **Windows** property.

```
ActiveWorkbook.Windows(1).Caption = "Consolidated Balance Sheet"  
ActiveWorkbook.Windows("Consolidated Balance Sheet") _  
    .ActiveSheet.Calculate
```



# Category Property

Returns or sets the category for the specified name in the language of the macro. The name must refer to a custom function or command. Read/write **String**.

## Example

This example assumes that you created a custom function or command on a Microsoft Excel 4.0 macro sheet. The example displays the function category in the language of the macro. It assumes that the name of the custom function or command is the only name in the workbook.

```
With ActiveWorkbook.Names(1)
  If .MacroType <> xlNone Then
    MsgBox "The category for this name is " & .Category
  Else
    MsgBox "This name does not refer to" & _
      " a custom function or command."
  End If
End With
```



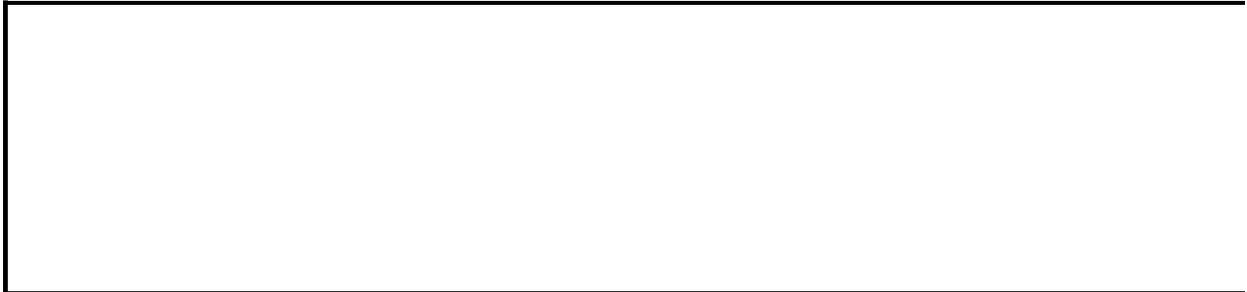
# CategoryLocal Property

Returns or sets the category for the specified name, in the language of the user, if the name refers to a custom function or command. Read/write **String**.

## Example

This example displays, in the language of the user, the function category of either a custom function or a command created on a Microsoft Excel 4.0 macro sheet. The example assumes that the custom function name or command name is the only name in the workbook.

```
With ActiveWorkbook.Names(1)
    If .MacroType <> xlNone Then
        MsgBox "The category for this name is " & .CategoryLocal
    Else
        MsgBox "This name does not refer to" & _
            " a custom function or command."
    End If
End With
```



# CategoryNames Property

Returns or sets all the category names for the specified axis, as a text array. When you set this property, you can set it to either an array or a **Range** object that contains the category names. Read/write **Variant**.

## Remarks

Category names are really a property of the "special" series in an axis grouping. Deleting or modifying that special series will change the category names for all series using the axis.

## Example

This example sets the category names for Chart1 to the values in cells B1:B5 on Sheet1.

```
Set Charts("Chart1").Axes(xlCategory).CategoryNames = _  
    Worksheets("Sheet1").Range("B1:B5")
```

This example uses an array to set individual category names for Chart1.

```
Charts("Chart1").Axes(xlCategory).CategoryNames = _  
    Array ("1985", "1986", "1987", "1988", "1989")
```



[Show All](#)

# CategoryType Property

Returns or sets the category axis type. Read/write [XlCategoryType](#).

XlCategoryType can be one of these XlCategoryType constants.

**xlCategoryScale**

**xlAutomaticScale**

**xlTimeScale**

*expression*.**CategoryType**

*expression* Required. An expression that returns one of the objects in the Applies To list.

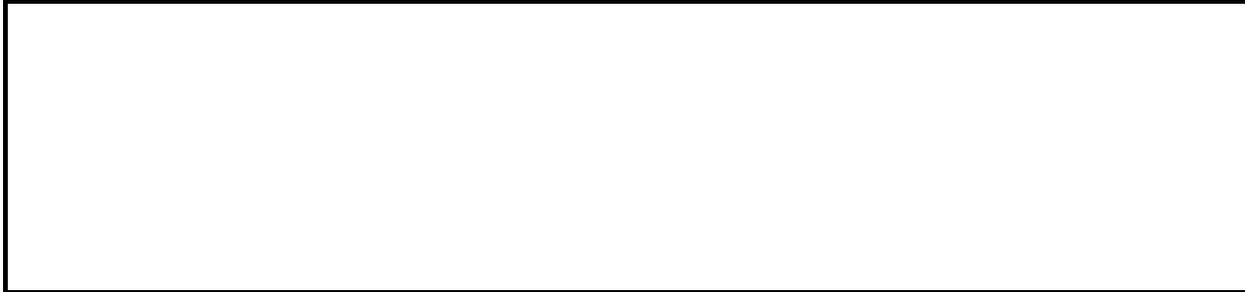
## Remarks

You cannot set this property for a value axis.

## Example

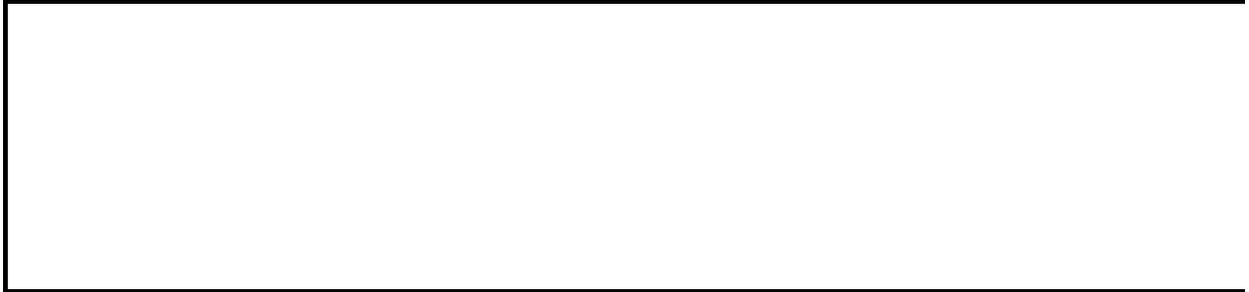
This example sets the category axis in embedded chart one on worksheet one to use a time scale, with months as the base unit.

```
With Worksheets(1).ChartObjects(1).Chart
    With .Axes(xlCategory)
        .CategoryType = xlTimeScale
        .BaseUnit = xlMonths
    End With
End With
```



# CCRecipients Property

You have requested Help for a Visual Basic keyword used only on the Macintosh. For information about this keyword, consult the language reference Help included with Microsoft Office Macintosh Edition.



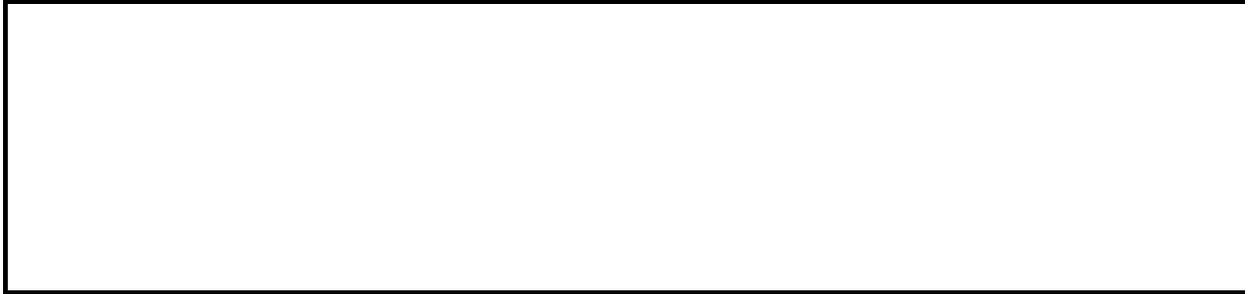
# CellDragAndDrop Property

**True** if dragging and dropping cells is enabled. Read/write **Boolean**.

## Example

This example enables dragging and dropping cells.

```
Application.CellDragAndDrop = True
```



[Show All](#)

# Cells Property

 [Cells Property as it applies to the \*\*Application\*\* object.](#)

Returns a **Range** object that represents all the cells on the active worksheet. If the active document isn't a worksheet, this property fails. Read-only.

*expression*.**Cells**

*expression* Required. An expression that returns an **Application** object.

 [Cells Property as it applies to the \*\*Range\*\* object.](#)

Returns a **Range** object that represents the cells in the specified range. Read-only.

*expression*.**Cells**

*expression* Required. An expression that returns a **Range** object.

 [Cells Property as it applies to the \*\*Worksheet\*\* object.](#)

Returns a **Range** object that represents all the cells on the worksheet (not just the cells that are currently in use). Read-only.

*expression*.**Cells**

*expression* Required. An expression that returns a **Worksheet** object.

## Remarks

Because the **Item** property is the [default property](#) for the **Range** object, you can specify the row and column index immediately after the **Cells** keyword. For more information, see the **Item** property and the examples for this topic.

Using this property without an object qualifier returns a **Range** object that represents all the cells on the active worksheet.

## Example

This example sets the font size for cell C5 on Sheet1 to 14 points.

```
Worksheets("Sheet1").Cells(5, 3).Font.Size = 14
```

This example clears the formula in cell one on Sheet1.

```
Worksheets("Sheet1").Cells(1).ClearContents
```

This example sets the font and font size for every cell on Sheet1 to 8-point Arial.

```
With Worksheets("Sheet1").Cells.Font
    .Name = "Arial"
    .Size = 8
End With
```

This example loops through cells A1:J4 on Sheet1. If a cell contains a value less than 0.001, the example replaces that value with 0 (zero).

```
For rwIndex = 1 to 4
    For colIndex = 1 to 10
        With Worksheets("Sheet1").Cells(rwIndex, colIndex)
            If .Value < .001 Then .Value = 0
        End With
    Next colIndex
Next rwIndex
```

This example sets the font style for cells A1:C5 on Sheet1 to italic.

```
Worksheets("Sheet1").Activate
Range(Cells(1, 1), Cells(5, 3)).Font.Italic = True
```

This example scans a column of data named "myRange." If a cell has the same value as the cell immediately above it, the example displays the address of the cell that contains the duplicate data.

```
Set r = Range("myRange")
```

```
For n = 1 To r.Rows.Count
    If r.Cells(n, 1) = r.Cells(n + 1, 1) Then
        MsgBox "Duplicate data in " & r.Cells(n + 1, 1).Address
    End If
Next n
```



# CenterFooter Property

Returns or sets the center part of the footer. Read/write **String**.

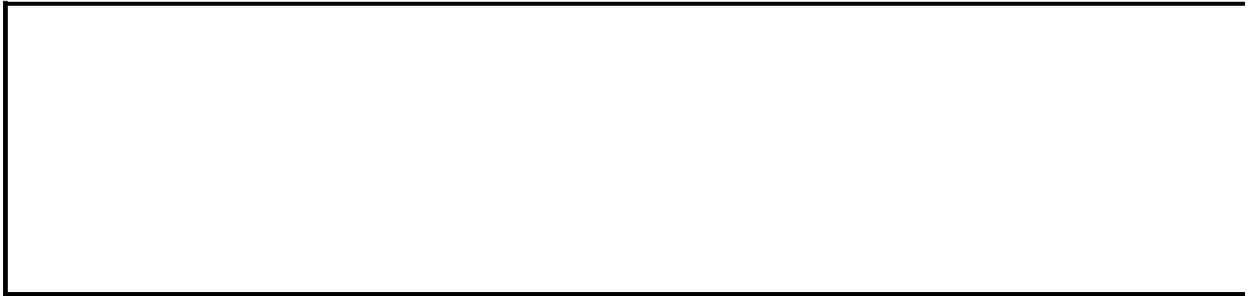
## Remarks

Special [format codes](#) can be used in the footer text.

## Example

This example prints the workbook name and page number at the bottom of each page.

```
Worksheets("Sheet1").PageSetup.CenterFooter = "&F page &P"
```



# CenterFooterPicture Property

Returns a [Graphic](#) object that represents the picture for the center section of the footer. Used to set attributes about the picture.

*expression*.**CenterFooterPicture**

*expression* Required. An expression that returns a [PageSetup](#) object.

## Remarks

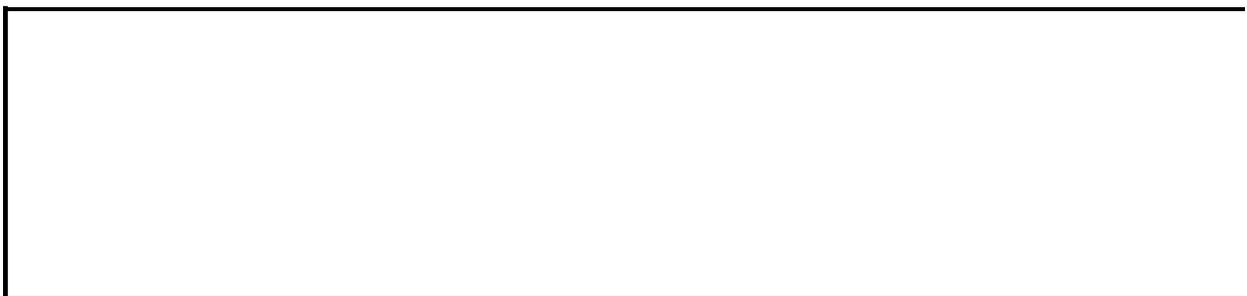
The **CenterFooterPicture** property is read-only, but the properties on it are not all read-only.

## Example

The following example adds a picture titled: Sample.jpg from the C:\ drive to the center section of the footer. This example assumes that a file called Sample.jpg exists on the C:\ drive.

```
Sub InsertPicture()  
  
    With ActiveSheet.PageSetup.CenterFooterPicture  
        .FileName = "C:\Sample.jpg"  
        .Height = 275.25  
        .Width = 463.5  
        .Brightness = 0.36  
        .ColorType = msoPictureGrayscale  
        .Contrast = 0.39  
        .CropBottom = -14.4  
        .CropLeft = -28.8  
        .CropRight = -14.4  
        .CropTop = 21.6  
    End With  
  
    ' Enable the image to show up in the center footer.  
    ActiveSheet.PageSetup.CenterFooter = "&G"  
  
End Sub
```

**Note** It is required that "&G" is a part of the **CenterFooter** property string in order for the image to show up in the center footer.



# CenterHeader Property

Returns or sets the center part of the header. Read/write **String**.

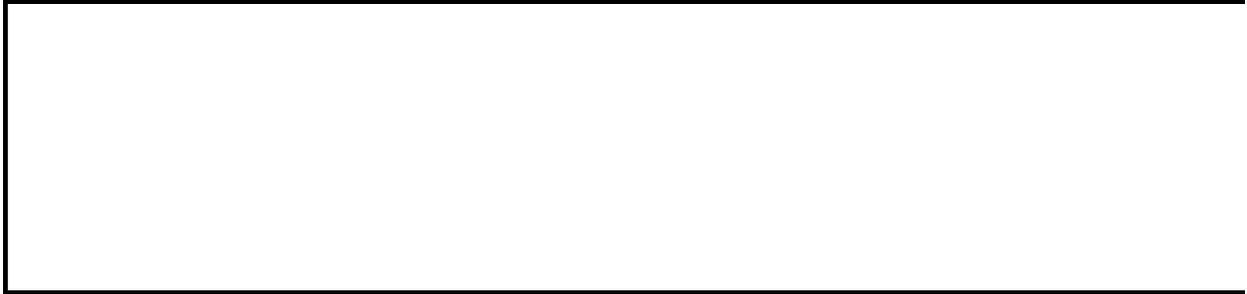
## Remarks

Special [format codes](#) can be used in the header text.

## Example

This example prints the date and page number at the top of each page.

```
Worksheets("Sheet1").PageSetup.CenterHeader = "&D page &P of &N"
```



# CenterHeaderPicture Property

Returns a [Graphic](#) object that represents the picture for the center section of the header. Used to set attributes about the picture.

*expression*.**CenterHeaderPicture**

*expression* Required. An expression that returns a [PageSetup](#) object.

## Remarks

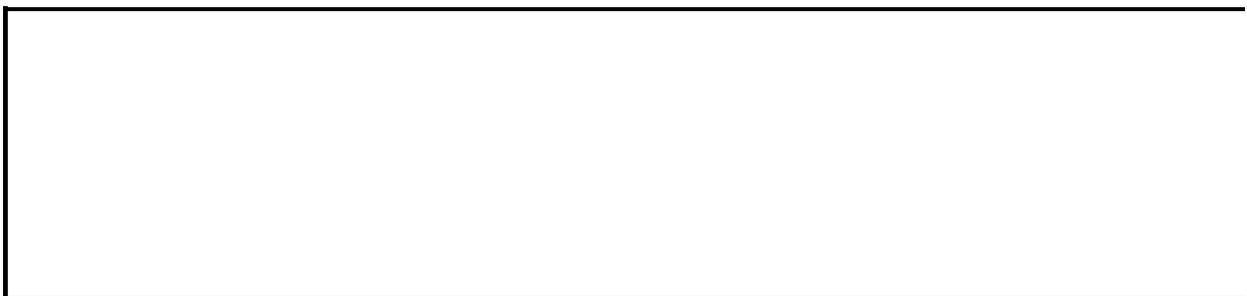
The **CenterHeaderPicture** property is read-only, but the properties on it are not all read-only.

## Example

The following example adds a picture titled: Sample.jpg from the C:\ drive to the center section of the header. This example assumes that a file called Sample.jpg exists on the C:\ drive.

```
Sub InsertPicture()  
  
    With ActiveSheet.PageSetup.CenterHeaderPicture  
        .FileName = "C:\Sample.jpg"  
        .Height = 275.25  
        .Width = 463.5  
        .Brightness = 0.36  
        .ColorType = msoPictureGrayscale  
        .Contrast = 0.39  
        .CropBottom = -14.4  
        .CropLeft = -28.8  
        .CropRight = -14.4  
        .CropTop = 21.6  
    End With  
  
    ' Enable the image to show up in the center header.  
    ActiveSheet.PageSetup.CenterHeader = "&G"  
  
End Sub
```

**Note** It is required that "&G" is a part of the **CenterHeader** property string in order for the image to show up in the center header.



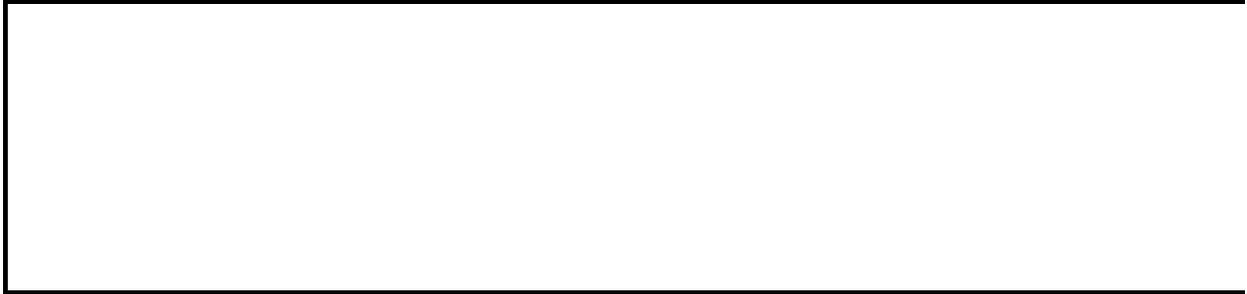
# CenterHorizontally Property

**True** if the sheet is centered horizontally on the page when it's printed.  
Read/write **Boolean**.

## Example

This example centers Sheet1 horizontally when it's printed.

```
Worksheets("Sheet1").PageSetup.CenterHorizontally = True
```



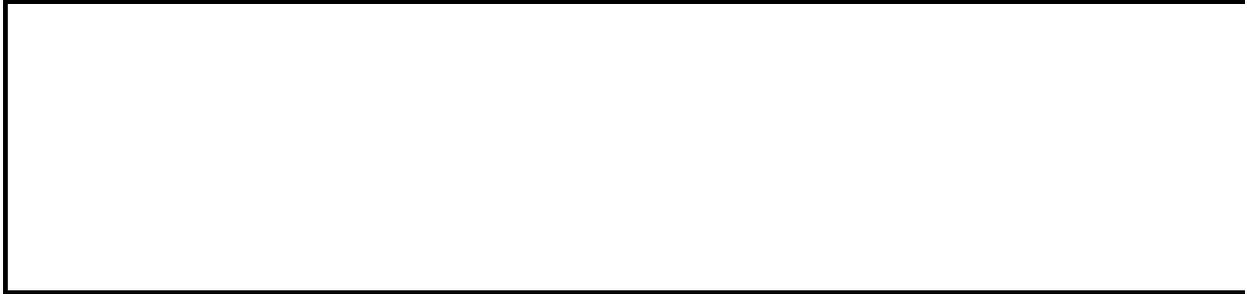
# CenterVertically Property

**True** if the sheet is centered vertically on the page when it's printed. Read/write **Boolean**.

## Example

This example centers Sheet1 vertically when it's printed.

```
Worksheets("Sheet1").PageSetup.CenterVertically = True
```



# ChangeHistoryDuration Property

Returns or sets the number of days shown in the shared workbook's change history. Read/write **Long**.

## **Remarks**

Any changes in the change history older than the setting for this property are removed when the workbook is closed.

## Example

This example sets the number of days shown in the change history for the active workbook if change tracking is enabled. Any changes in the change history older than the setting for this property are removed when the workbook is closed.

```
With ActiveWorkbook
    If .KeepChangeHistory Then
        .ChangeHistoryDuration = 7
    End If
End With
```



# ChangingCells Property

Returns a [Range](#) object that represents the changing cells for a scenario. Read-only.

## Example

This example selects the changing cells for scenario one on Sheet1.

```
Worksheets("Sheet1").Activate  
ActiveSheet.Scenarios(1).ChangingCells.Select
```



# Characters Property

Returns a [Characters](#) object that represents a range of characters within the object text. You can use the **Characters** object to format characters within a text string.

*expression*.**Characters**(*Start*, *Length*)

*expression* Required. An expression that returns an object in the Applies To list.

**Start** Optional **Variant**. The first character to be returned. If this argument is either 1 or omitted, this property returns a range of characters starting with the first character.

**Length** Optional **Variant**. The number of characters to be returned. If this argument is omitted, this property returns the remainder of the string (everything after the **Start** character).

## Remarks

The **Characters** object isn't a collection.

For the [TextFrame](#) object, [Characters](#) is a method.

## Example

This example formats the third character in cell A1 on Sheet1 as bold.

```
With Worksheets("Sheet1").Range("A1")  
    .Value = "abcdefg"  
    .Characters(3, 1).Font.Bold = True  
End With
```



[Show All](#)

# CharacterType Property

Returns or sets the type of phonetic text in the specified cell. Read/write [XlPhoneticCharacterType](#).

XlPhoneticCharacterType can be one of these XlPhoneticCharacterType constants.

**xlHiragana**

**xlKatakana**

**xlKatakanaHalf**

**xlNoConversion**

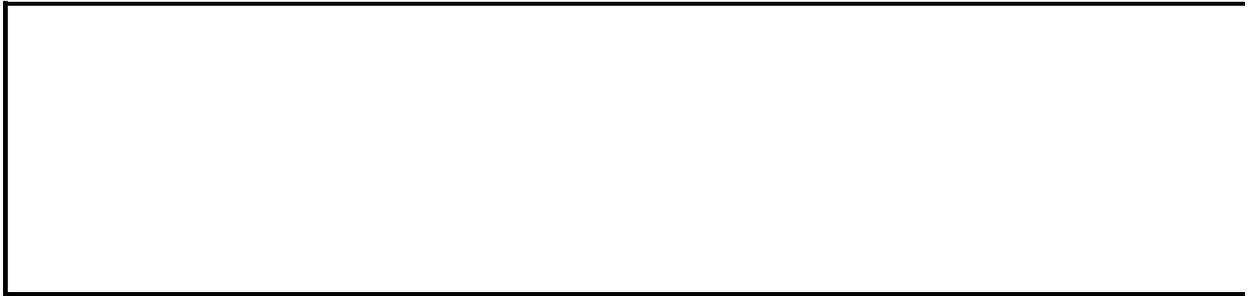
*expression*.**CharacterType**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example changes the first phonetic text string in the active cell from Furigana to Hiragana.

```
ActiveCell.Phonetics(1).CharacterType = xlHiragana
```



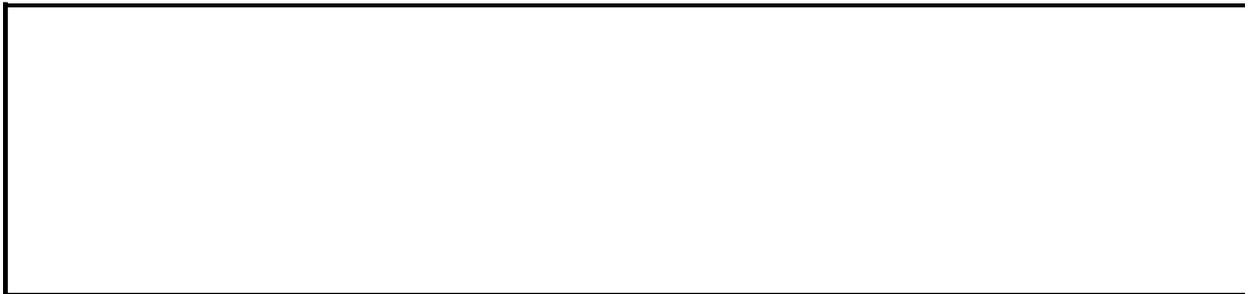
# Chart Property

Returns a [Chart](#) object that represents the chart contained in the object. Read-only.

## Example

This example adds a title to the first embedded chart on Sheet1.

```
With Worksheets("Sheet1").ChartObjects(1).Chart
    .HasTitle = True
    .ChartTitle.Text = "1995 Rainfall Totals by Month"
End With
```



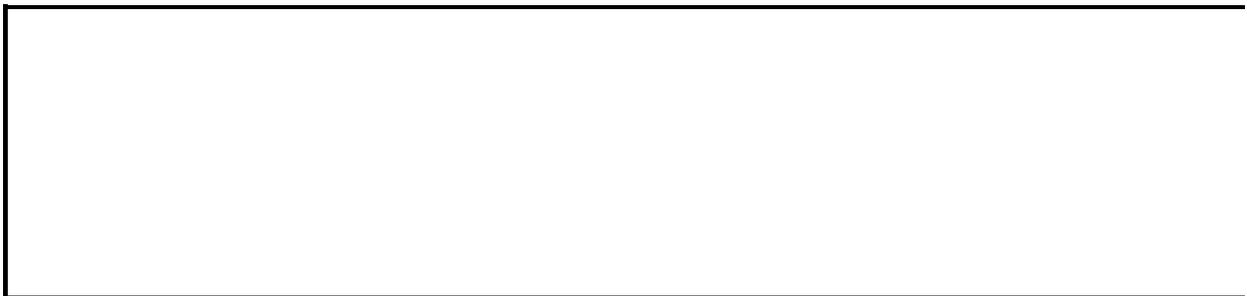
# ChartArea Property

Returns a [ChartArea](#) object that represents the complete chart area for the chart.  
Read-only.

## Example

This example sets the chart area interior color of Chart1 to red and sets the border color to blue.

```
With Charts("Chart1").ChartArea  
    .Interior.ColorIndex = 3  
    .Border.ColorIndex = 5  
End With
```



[Show All](#)

# Charts Property

 [Charts Property as it applies to the \*\*Application\*\* object.](#)

Returns a **Sheets** collection that represents all the chart sheets in the active workbook. Read-only.

*expression*.**Charts**

*expression* Required. An expression that returns an object in the Applies To List.

Using this property without an object qualifier returns all chart sheets in the active workbook.

For information about returning a single member of a collection, see [Returning an Object from a Collection](#).

 [Charts Property as it applies to the \*\*Workbook\*\* object.](#)

Returns a **Sheets** collection that represents all the chart sheets in the specified workbook. Read-only.

*expression*.**Charts**

*expression* Required. An expression that returns an object in the Applies To List.

Using this property without an object qualifier returns all chart sheets in the active workbook.

For information about returning a single member of a collection, see [Returning an Object from a Collection](#).

## Example

This example sets the text for the title of Chart1.

```
With Charts("Chart1")  
    .HasTitle = True  
    .ChartTitle.Text = "First Quarter Sales"  
End With
```

This example deletes every chart sheet in the active workbook.

```
ActiveWorkbook.Charts.Delete
```

This example hides Chart1, Chart3, and Chart5.

```
Charts(Array("Chart1", "Chart3", "Chart5")).Visible = False
```



[Show All](#)

# ChartSize Property

Returns or sets the way a chart is scaled to fit on a page. Read/write [XlObjectSize](#).

XlObjectSize can be one of these XlObjectSize constants.

**xlFitToPage.** Print the chart as large as possible, while retaining the chart's height-to-width ratio as shown on the screen

**xlFullPage.** Print the chart to fit the page, adjusting the height-to-width ratio as necessary.

**xlScreenSize.** Print the chart the same size as it appears on the screen.

*expression*.**ChartSize**

*expression* Required. An expression that returns one of the objects in the Applies To list.

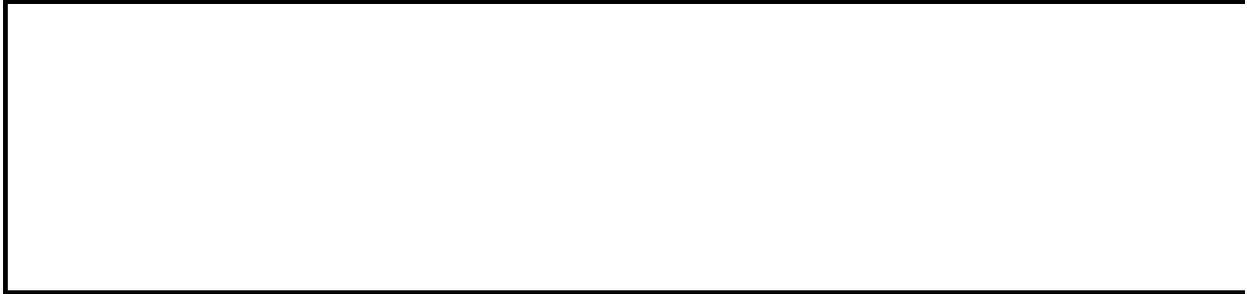
## Remarks

This property applies only to chart sheets (it cannot be used with embedded charts).

## Example

This example scales the first chart in the active workbook to fit a full page.

```
ActiveWorkbook.Charts(1).PageSetup.ChartSize = xlFullPage
```



# ChartTitle Property

Returns a [ChartTitle](#) object that represents the title of the specified chart. Read-only.

## Example

This example sets the text for the title of Chart1.

```
With Charts("Chart1")  
    .HasTitle = True  
    .ChartTitle.Text = "First Quarter Sales"  
End With
```



[Show All](#)

# ChartType Property

Returns or sets the chart type. Read/write [XLChartType](#).

XLChartType can be one of these XLChartType constants.

**xlLine**. Line

**xlLineMarkersStacked**. Stacked Line with Markers

**xlLineStacked**. Stacked Line

**xlPie**. Pie

**xlPieOfPie**. Pie of Pie

**xlPyramidBarStacked**. Stacked Pyramid Bar

**xlPyramidCol**. 3D Pyramid Column

**xlPyramidColClustered**. Clustered Pyramid Column

**xlPyramidColStacked**. Stacked Pyramid Column

**xlPyramidColStacked100**. 100% Stacked Pyramid Column

**xlRadar**. Radar

**xlRadarFilled**. Filled Radar

**xlRadarMarkers**. Radar with Data Markers

**xlStockHLC**. High-Low-Close

**xlStockOHLC**. Open-High-Low-Close

**xlStockVHLC**. Volume-High-Low-Close

**xlStockVOHLC**. Volume-Open-High-Low-Close

**xlSurface**. 3D Surface

**xlSurfaceTopView**. Surface (Top View)

**xlSurfaceTopViewWireframe**. Surface (Top View wireframe)

**xlSurfaceWireframe**. 3D Surface (wireframe)

**xlXYScatter**. Scatter

**xlXYScatterLines**. Scatter with Lines.

**xlXYScatterLinesNoMarkers**. Scatter with Lines and No Data Markers

**xlXYScatterSmooth**. Scatter with Smoothed Lines

**xlXYScatterSmoothNoMarkers**. Scatter with Smoothed Lines and No Data Markers

**xl3DArea.** 3D Area  
**xl3DAreaStacked.** 3D Stacked Area  
**xl3DAreaStacked100.** 100% Stacked Area  
**xl3DBarClustered.** 3D Clustered Bar  
**xl3DBarStacked.** 3D Stacked Bar  
**xl3DBarStacked100.** 3D 100% Stacked Bar  
**xl3DColumn.** 3D Column  
**xl3DColumnClustered.** 3D Clustered Column  
**xl3DColumnStacked.** 3D Stacked Column  
**xl3DColumnStacked100.** 3D 100% Stacked Column  
**xl3DLine.** 3D Line  
**xl3DPie.** 3D Pie  
**xl3DPieExploded.** Exploded 3D Pie  
**xlArea.** Area  
**xlAreaStacked.** Stacked Area  
**xlAreaStacked100.** 100% Stacked Area  
**xlBarClustered.** Clustered Bar  
**xlBarOfPie.** Bar of Pie  
**xlBarStacked.** Stacked Bar  
**xlBarStacked100.** 100% Stacked Bar  
**xlBubble.** Bubble  
**xlBubble3DEffect.** Bubble with 3D effects  
**xlColumnClustered.** Clustered Column  
**xlColumnStacked.** Stacked Column  
**xlColumnStacked100.** 100% Stacked Column  
**xlConeBarClustered.** Clustered Cone Bar  
**xlConeBarStacked.** Stacked Cone Bar  
**xlConeBarStacked100.** 100% Stacked Cone Bar  
**xlConeCol.** 3D Cone Column  
**xlConeColClustered.** Clustered Cone Column  
**xlConeColStacked.** Stacked Cone Column  
**xlConeColStacked100.** 100% Stacked Cone Column  
**xlCylinderBarClustered.** Clustered Cylinder Bar

**xlCylinderBarStacked.** Stacked Cylinder Bar  
**xlCylinderBarStacked100.** 100% Stacked Cylinder Bar  
**xlCylinderCol.** 3D Cylinder Column  
**xlCylinderColClustered.** Clustered Cone Column  
**xlCylinderColStacked.** Stacked Cone Column  
**xlCylinderColStacked100.** 100% Stacked Cylinder Column  
**xlDoughnut.** Doughnut  
**xlDoughnutExploded.** Exploded Doughnut  
**xlLineMarkers.** Line with Markers  
**xlLineMarkersStacked100.** 100% Stacked Line with Markers  
**xlLineStacked100.** 100% Stacked Line  
**xlPieExploded.** Exploded Pie  
**xlPyramidBarClustered.** Clustered Pyramid Bar  
**xlPyramidBarStacked100.** 100% Stacked Pyramid Bar

*expression*.**ChartType**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## **Remarks**

Some chart types aren't available for PivotChart reports.

## Example

This example sets the bubble size in chart group one to 200% of the default size if the chart is a 2D bubble chart.

```
With Worksheets(1).ChartObjects(1).Chart
    If .ChartType = xlBubble Then
        .ChartGroups(1).BubbleScale = 200
    End If
End With
```



# CheckboxState Property

Returns or sets a **Boolean** value that indicates whether a check box in a smart document is selected. Setting this property to **True** selects the check box. Setting this property to **False** clears the check box. Read/write **Boolean**.

*expression*.**CheckboxState**

*expression* Required. An expression that returns a **SmartTagAction** object.

## Remarks

For more information on smart documents, see the Smart Document Software Development Kit on the [Microsoft Developer Network \(MSDN\)](#) Web site.

---

---

---

# CheckIfOfficeIsHTMLEditor Property

**True** if Microsoft Excel checks to see whether an Office application is the default HTML editor when you start Excel. **False** if Excel does not perform this check. The default value is **True**. Read/write **Boolean**.

## Remarks

This property is used only if the Web browser you are using supports HTML editing and HTML editors.

To use a different HTML editor, you must set this property to **False** and then register the editor as the default system HTML editor.

## Example

This example causes Microsoft Excel not to check to see whether it is the default HTML editor.

```
Application.DefaultWebOptions.CheckIfOfficeIsHTMLEditor = False
```



[Show All](#)

# Child Property

Returns **msoTrue** if the specified shape is a child shape or if all shapes in a shape range are child shapes of the same parent. Read-only [MsoTriState](#).

MsoTriState can be one of these MsoTriState constants.

**msoCTrue** Does not apply to this property.

**msoFalse** If the selected shape is not a child shape.

**msoTriStateMixed** If only some of the selected shapes are child shapes.

**msoTriStateToggle** Does not apply to this property.

**msoTrue** If the selected shape is a child shape.

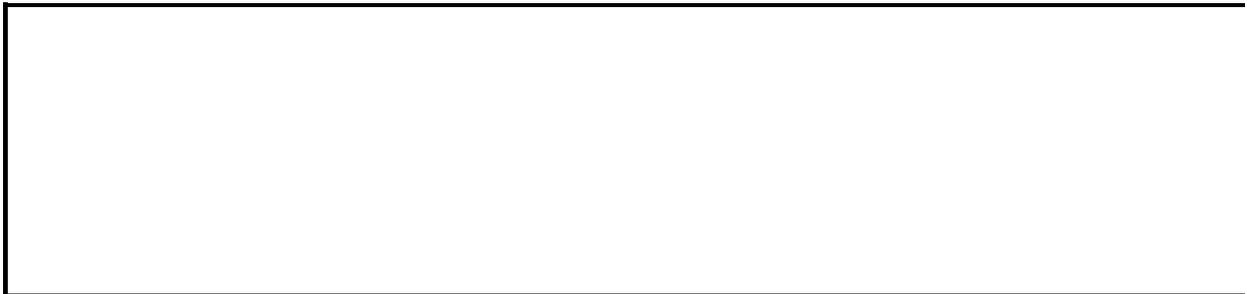
*expression*.**Child**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example selects the first shape in the canvas, and if the selected shape is a child shape, fills the shape with the specified color. This example assumes that a drawing canvas contains multiple shapes on the active worksheet.

```
Sub FillChildShape()  
  
    'Select the first shape in the drawing canvas.  
    ActiveSheet.Shapes(1).CanvasItems(1).Select  
  
    'Fill selected shape if it is a child shape.  
    If Selection.ShapeRange.Child = msoTrue Then  
        Selection.ShapeRange.Fill.ForeColor.RGB = RGB(100, 0, 200)  
    Else  
        MsgBox "This shape is not a child shape."  
    End If  
  
End Sub
```



[Show All](#)

# ChildField Property

Returns a [PivotField](#) object that represents the child field for the specified field (if the field is grouped and has a child field). Read-only.

## Remarks

If the specified field has no child field, this property causes an error.

This property is not available for [OLAP](#) data sources.

## Example

This example displays the name of the child field for the field named "REGION2."

```
Set pvtTable = Worksheets("Sheet1").Range("A3").PivotTable
MsgBox "The name of the child field is " & _
    pvtTable.PivotFields("REGION2").ChildField.Name
```



[Show All](#)

# ChildItems Property

Returns an object that represents either a single PivotTable item (a [PivotItem](#) object) or a collection of all the items (a [PivotItems](#) object) that are group children in the specified field, or children of the specified item. Read-only.

*expression*.**ChildItems**(*Index*)

*expression* Required. An expression that returns a **PivotField** or **PivotItem** object.

**Index** Optional **Variant**. The item name or number (can be an array to specify more than one item).

## Remarks

This property is not available for [OLAP](#) data sources.

## Example

This example adds the names of all the child items of the item named "vegetables" to a list on a new worksheet.

```
Set nwSheet = Worksheets.Add
nwSheet.Activate
Set pvtTable = Worksheets("Sheet2").Range("A1").PivotTable
rw = 0
For Each pvtItem In _
    pvtTable.PivotFields("product")
        .PivotItems("vegetables").ChildItems
    rw = rw + 1
    nwSheet.Cells(rw, 1).Value = pvtItem.Name
Next pvtItem
```



# Children Property

Returns a [DiagramNodeChildren](#) object, representing the collection of child nodes of a particular node.

*expression*.**Children**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

The following example creates a diagram and adds child nodes to it.

```
Sub CreatePyramidDiagram()  
  
    Dim dgnNode As DiagramNode  
    Dim shpDiagram As Shape  
    Dim intCount As Integer  
  
    'Add pyramid diagram to current document.  
    Set shpDiagram = ActiveSheet.Shapes.AddDiagram( _  
        Type:=msoDiagramPyramid, Left:=10, _  
        Top:=15, Width:=400, Height:=475)  
  
    'Add first child diagram node.  
    Set dgnNode = shpDiagram.DiagramNode.Children.AddNode  
  
    'Add three more nodes.  
    For intCount = 1 To 3  
        dgnNode.AddNode  
    Next intCount  
  
End Sub
```



[Show All](#)

# Choices Property

Returns an **Array** of **String** values that contains the choices offered to the user by the **ListLookUp**, **ChoiceMulti**, and **Choice** data types of the **DefaultValue** property. Read-only **Variant**.

*expression*. **Choices**

*expression* Required. An expression that returns a **ListDataFormat** object.

## Remarks

In Microsoft Excel, you cannot set any of the properties associated with the **ListDataFormat** object. You can set these properties, however, by modifying the list on the server that is running Microsoft Windows SharePoint Services.

## Example

The following example displays the setting of the **Choice** property for the third column in a [list](#) that is linked to a SharePoint list. In this example, it is assumed that the **DefaultValue** property has been set to the **Choice**, **ChoiceMulti**, or **ListLookup** data type.

```
Sub PrintChoices()  
    Dim wrksht As Worksheet  
    Dim objListCol As ListColumn  
  
    Set wrksht = ActiveWorkbook.Worksheets("Sheet1")  
    Set objListCol = wrksht.ListObjects(1).ListColumns(3)  
  
    Debug.Print objListCol.ListDataFormat.Choices  
End Sub
```



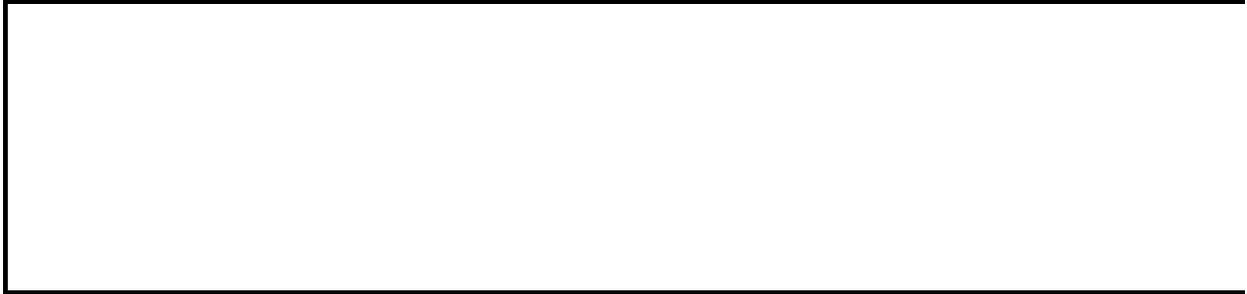
# CircularReference Property

Returns a [Range](#) object that represents the range containing the first circular reference on the sheet, or returns **Nothing** if there's no circular reference on the sheet. The circular reference must be removed before calculation can proceed. Read-only.

## Example

This example selects the first cell in the first circular reference on Sheet1.

```
worksheets("Sheet1").CircularReference.Select
```



[Show All](#)

# ClipboardFormats Property

Returns the formats that are currently on the Clipboard, as an array of numeric values. To determine whether a particular format is on the Clipboard, compare each element in the array with the appropriate constant listed in the Remarks section. Read-only **Variant**.

*expression*.**ClipboardFormats**(*Index*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Index** Optional **Variant**. The array element to be returned. If this argument is omitted, the property returns the entire array of formats that are currently on the Clipboard. For more information, see the Remarks section.

## Remarks

This property returns an array of numeric values. To determine whether a particular format is on the Clipboard compare each element of the array with one of the following [xlClipboardFormat](#) constants:

<b>xlClipboardFormatBIFF</b>	<b>xlClipboardFormatObjectDesc</b>
<b>xlClipboardFormatBIFF2</b>	<b>xlClipboardFormatObjectLink</b>
<b>xlClipboardFormatBIFF3</b>	<b>xlClipboardFormatOwnerLink</b>
<b>xlClipboardFormatBIFF4</b>	<b>xlClipboardFormatPICT</b>
<b>xlClipboardFormatBinary</b>	<b>xlClipboardFormatPrintPICT</b>
<b>xlClipboardFormatBitmap</b>	<b>xlClipboardFormatRTF</b>
<b>xlClipboardFormatCGM</b>	<b>xlClipboardFormatScreenPICT</b>
<b>xlClipboardFormatCSV</b>	<b>xlClipboardFormatStandardFont</b>
<b>xlClipboardFormatDIF</b>	<b>xlClipboardFormatStandardScale</b>
<b>xlClipboardFormatDspText</b>	<b>xlClipboardFormatSYLK</b>
<b>xlClipboardFormatEmbeddedObject</b>	<b>xlClipboardFormatTable</b>
<b>xlClipboardFormatEmbedSource</b>	<b>xlClipboardFormatText</b>
<b>xlClipboardFormatLink</b>	<b>xlClipboardFormatToolFace</b>
<b>xlClipboardFormatLinkSource</b>	<b>xlClipboardFormatToolFacePICT</b>
<b>xlClipboardFormatLinkSourceDesc</b>	<b>xlClipboardFormatVALU</b>
<b>xlClipboardFormatMovie</b>	<b>xlClipboardFormatWK1</b>
<b>xlClipboardFormatNative</b>	

## Example

This example displays a message box if the Clipboard contains a rich-text format (RTF) object. You can create an RTF object by copying text from a Word document.

```
aFmts = Application.ClipboardFormats  
For Each fmt In aFmts  
    If fmt = xlClipboardFormatRTF Then  
        MsgBox "Clipboard contains rich text"  
    End If  
Next
```



# CLSID Property

Returns a read-only unique identifier, or CLSID, identifying an object, as a **String**.

*expression*.**CLSID**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example returns the CLSID of an add-in titled "Analysis ToolPak". This example assumes the "Analysis ToolPak" has been installed.

```
Sub FindCLSID()
```

```
    MsgBox Application.AddIns("Analysis ToolPak").CLSID
```

```
End Sub
```



# CodeName Property

Returns the code name for the object. Read-only **String**.

**Note** The value that you see in the cell to the right of **(Name)** in the **Properties** window is the code name of the selected object. At design time, you can change the code name of an object by changing this value. You cannot programmatically change this property at run time.

## Remarks

The code name for an object can be used in place of an expression that returns the object. For example, if the code name for worksheet one is "Sheet1", the following expressions are identical:

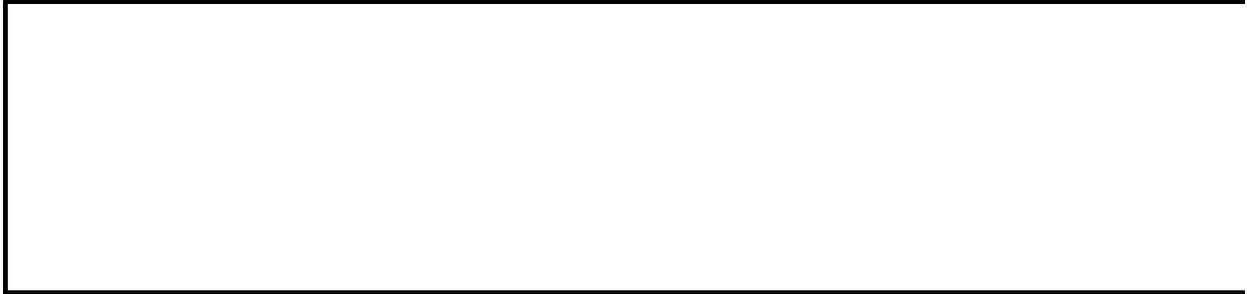
```
Worksheets(1).Range("a1")  
Sheet1.Range("a1")
```

It's possible for the sheet name to be different from the code name. When you create a sheet, the sheet name and code name are the same, but changing the sheet name doesn't change the code name, and changing the code name (using the **Properties** window in the Visual Basic Editor) doesn't change the sheet name.

## Example

This example displays the code name for worksheet one.

```
MsgBox Worksheets(1).CodeName
```



# Color Property

Returns or sets the primary color of the object, as shown in the following table. Use the **RGB** function to create a color value. Read/write **Variant**.

<b>Object</b>	<b>Color</b>
<b>Border</b>	The color of the border.
<b>Borders</b>	The color of all four borders of a range. If they're not all the same color, <b>Color</b> returns 0 (zero).
<b>Font</b>	The color of the font.
<b>Interior</b>	The cell shading color or the drawing object fill color.

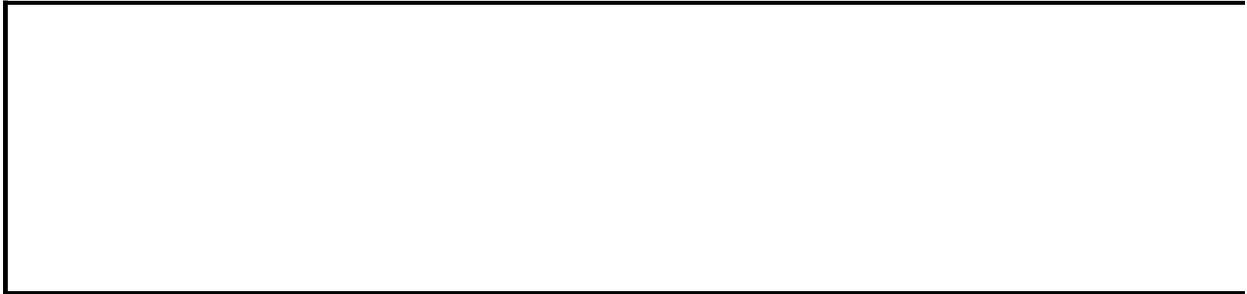
*expression*.**Color**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example sets the color of the tick-mark labels on the value axis in Chart1.

```
Charts("Chart1").Axes(x1Value).TickLabels.Font.Color = _  
    RGB(0, 255, 0)
```



[Show All](#)

# ColorIndex Property

 [ColorIndex Property as it applies to the \*\*Border\*\* object.](#)

Returns or sets the color of the border. The color is specified as an index value into the current color palette, or as one of the following [XIColorIndex](#) constants. Read/write **Variant**.

XIColorIndex can be one of these XIColorIndex constants.

**xIColorIndexAutomatic**

**xIColorIndexNone**

*expression*.**ColorIndex**

*expression* Required. An expression that returns an object in the Applies To List.

 [ColorIndex Property as it applies to the \*\*Borders\*\* object.](#)

Returns or sets the color of all four borders. Returns **Null** if all four borders aren't the same color. The color is specified as an index value into the current color palette, or as one of the following [XIColorIndex](#) constants. Read/write **Variant**.

XIColorIndex can be one of these XIColorIndex constants.

**xIColorIndexAutomatic**

**xIColorIndexNone**

*expression*.**ColorIndex**

*expression* Required. An expression that returns an object in the Applies To List.

 [ColorIndex Property as it applies to the \*\*Font\*\* object.](#)

Returns or sets the color of the font. The color is specified as an index value into the current color palette, or as one of the following [XIColorIndex](#) constants.  
Read/write **Variant**.

XIColorIndex can be one of these XIColorIndex constants.

**xIColorIndexAutomatic**. Use to specify automatic color.

**xIColorIndexNone**.

*expression*.**ColorIndex**

*expression* Required. An expression that returns an object in the Applies To List.

 [ColorIndex Property as it applies to the Interior object.](#)

Returns or sets the color of the interior. The color is specified as an index value into the current color palette, or as one of the following [XIColorIndex](#) constants.  
Read/write **Variant**.

XIColorIndex can be one of these XIColorIndex constants.

**xIColorIndexAutomatic**. Use to specify the automatic fill, for drawing objects.

**xIColorIndexNone**. Use to specify no interior fill.

*expression*.**ColorIndex**

*expression* Required. An expression that returns an object in the Applies To List.

## Example

The following examples assume that you're using the default color palette.

 [Example as it applies to the \*\*Border\*\* object.](#)

This example sets the color of the major gridlines for the value axis in Chart1.

```
With Charts("Chart1").Axes(xlValue)
    If .HasMajorGridlines Then
        .MajorGridlines.Border.ColorIndex = 5      'set color to blue
    End If
End With
```

 [Example as it applies to the \*\*Font\*\* object.](#)

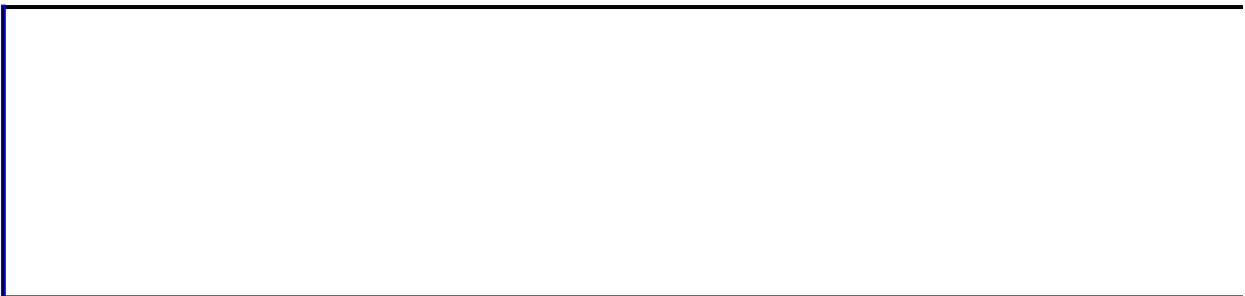
This example changes the font color in cell A1 on Sheet1 to red.

```
Worksheets("Sheet1").Range("A1").Font.ColorIndex = 3
```

 [Example as it applies to the \*\*Interior and Border\*\* objects.](#)

This example sets the color of the chart area interior of Chart1 to red and sets the border color to blue.

```
With Charts("Chart1").ChartArea
    .Interior.ColorIndex = 3
    .Border.ColorIndex = 5
End With
```



# Colors Property

Returns or sets colors in the palette for the workbook. The palette has 56 entries, each represented by an RGB value. Read/write **Variant**.

*expression*.**Colors**(*Index*)

*expression* Required. An expression that returns a **Workbook** object.

*Index* Optional **Variant**. The color number (from 1 to 56). If this argument isn't specified, this method returns an array that contains all 56 of the colors in the palette.

## Example

This example sets the color palette for the active workbook to be the same as the palette for Book2.xls.

```
ActiveWorkbook.Colors = Workbooks("BOOK2.XLS").Colors
```

This example sets color five in the color palette for the active workbook.

```
ActiveWorkbook.Colors(5) = RGB(255, 0, 0)
```



[Show All](#)

# ColorType Property

Returns or sets the type of color transformation applied to the specified picture or OLE object. Read/write [MsoPictureColorType](#).

MsoPictureColorType can be one of these MsoPictureColorType constants.

**msoPictureAutomatic**

**msoPictureBlackAndWhite**

**msoPictureGrayscale**

**msoPictureMixed**

**msoPictureWatermark**

*expression*.**ColorType**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example sets the color transformation to grayscale for shape one on myDocument. Shape one must be either a picture or an OLE object.

```
Set myDocument = Worksheets(1)  
myDocument.Shapes(1).PictureFormat.ColorType = msoPictureGrayscale
```



# Column Property

Returns the number of the first column in the first area in the specified range.  
Read-only **Long**.

## Remarks

Column A returns 1, column B returns 2, and so on.

To return the number of the last column in the range, use the following expression.

```
myRange.Columns(myRange.Columns.Count).Column
```

## Example

This example sets the column width of every other column on Sheet1 to 4 points.

```
For Each col In Worksheets("Sheet1").Columns
    If col.Column Mod 2 = 0 Then
        col.ColumnWidth = 4
    End If
Next col
```



# Column3DGroup Property

Returns a [ChartGroup](#) object that represents the column chart group on a 3-D chart. Read-only.

## Example

This example sets the space between column clusters in the 3-D column chart group to be 50 percent of the column width.

```
Charts(1).Column3DGroup.GapWidth = 50
```



# ColumnFields Property

Returns an object that represents either a single PivotTable field (a [PivotField](#) object) or a collection of all the fields (a [PivotFields](#) object) that are currently shown as column fields. Read-only.

*expression*.**ColumnFields**(*Index*)

*expression* Required. An expression that returns a **PivotTable** object.

**Index** Optional **Variant**. The field name or number (can be an array to specify more than one field).

## Example

This example adds the field names of the PivotTable report columns to a list on a new worksheet.

```
Set nwSheet = Worksheets.Add  
nwSheet.Activate  
Set pvtTable = Worksheets("Sheet2").Range("A1").PivotTable  
rw = 0  
For Each pvtField In pvtTable.ColumnFields  
    rw = rw + 1  
    nwSheet.Cells(rw, 1).Value = pvtField.Name  
Next pvtField
```



# ColumnGrand Property

**True** if the PivotTable report shows grand totals for columns. Read/write **Boolean**.

## Example

This example sets the PivotTable report to show grand totals for columns.

```
Set pvtTable = Worksheets("Sheet1").Range("A3").PivotTable  
pvtTable.ColumnGrand = True
```



# ColumnItems Property

Returns a [PivotItemList](#) collection that corresponds to the items on the column axis that represent the selected range.

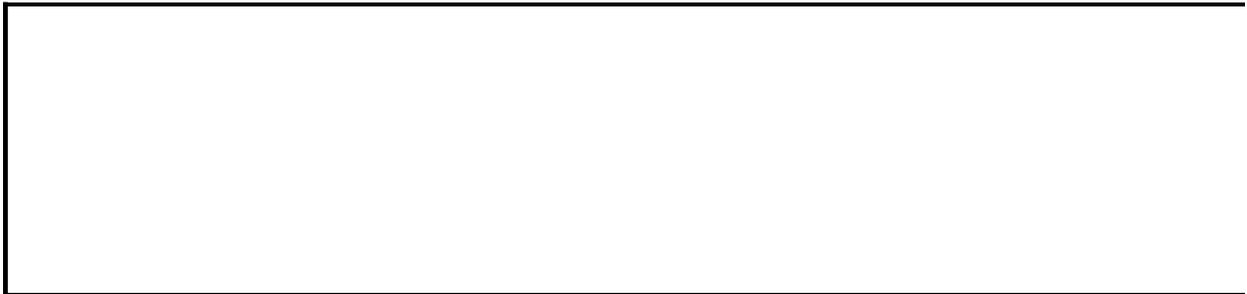
*expression*.**ColumnItems**

*expression* Required. An expression that returns a [PivotCell](#) object.

## Example

This example determines if the data item in cell B5 is under the Inventory item in the first column field and notifies the user. The example assumes that a PivotTable exists on the active worksheet and that column B contains a column field of the PivotTable.

```
Sub CheckColumnItems()  
    ' Determine if there is a match between the item and column field  
    If Application.Range("B5").PivotCell.ColumnItems.Item(1) = "Inventory"  
        MsgBox "Item in B5 is a member of the 'Inventory' column field"  
    Else  
        MsgBox "Item in B5 is not a member of the 'Inventory' column field"  
    End If  
End Sub
```



# ColumnRange Property

Returns a [Range](#) object that represents the range that contains the column area in the PivotTable report. Read-only.

## Example

This example selects the column headers for the PivotTable report.

```
Worksheets("Sheet1").Activate  
Range("A3").Select  
ActiveCell.PivotTable.ColumnRange.Select
```



[Show All](#)

# Columns Property

 [Columns property as it applies to the \*\*Application\*\* object.](#)

Returns a **Range** object that represents all the columns on the active worksheet. If the active document isn't a worksheet, the **Columns** property fails. Read-only.

*expression*.**Columns**

*expression* Required. An expression that returns an object in the Applies To List.

 [Columns property as it applies to the \*\*Range\*\* object.](#)

Returns a **Range** object that represents the columns in the specified range. Read-only.

*expression*.**Columns**

*expression* Required. An expression that returns an object in the Applies To List.

 [Columns property as it applies to the \*\*WorkSheet\*\* object.](#)

Returns a **Range** object that represents all the columns on the specified worksheet. Read-only.

*expression*.**Columns**

*expression* Required. An expression that returns an object in the Applies To List.

For information about returning a single member of a collection, see [Returning an Object from a Collection](#).

## Remarks

Using this property without an object qualifier is equivalent to using `ActiveSheet.Columns`.

When applied to a **Range** object that's a multiple-area selection, this property returns columns from only the first area of the range. For example, if the **Range** object has two areas— `A1:B2` and `C3:D4`— `Selection.Columns.Count` returns 2, not 4. To use this property on a range that may contain a multiple-area selection, test `Areas.Count` to determine whether the range contains more than one area. If it does, loop over each area in the range.

## Example

This example formats the font of column one (column A) on Sheet1 as bold.

```
Worksheets("Sheet1").Columns(1).Font.Bold = True
```

This example sets the value of every cell in column one in the range named "myRange" to 0 (zero).

```
Range("myRange").Columns(1).Value = 0
```

This example displays the number of columns in the selection on Sheet1. If more than one area is selected, the example loops through each area.

```
Worksheets("Sheet1").Activate
areaCount = Selection.Areas.Count
If areaCount <= 1 Then
    MsgBox "The selection contains " & _
        Selection.Columns.Count & " columns."
Else
    For i = 1 To areaCount
        MsgBox "Area " & i & " of the selection contains " & _
            Selection.Areas(i).Columns.Count & " columns."
    Next i
End If
```



# ColumnWidth Property

Returns or sets the width of all columns in the specified range. Read/write **Variant**.

## Remarks

One unit of column width is equal to the width of one character in the Normal style. For proportional fonts, the width of the character 0 (zero) is used.

Use the [Width](#) property to return the width of a column in points.

If all columns in the range have the same width, the **ColumnWidth** property returns the width. If columns in the range have different widths, this property returns **Null**.

## Example

This example doubles the width of column A on Sheet1.

```
With Worksheets("Sheet1").Columns("A")  
    .ColumnWidth = .ColumnWidth * 2  
End With
```



# COMAddIns Property

Returns the [COMAddIns](#) collection for Microsoft Excel, which represents the currently installed COM add-ins. Read-only.

## Example

This example displays the number of COM add-ins that are currently installed.

```
Set objAI = Application.COMAddIns  
MsgBox "Number of COM add-ins available:" & _  
    objAI.Count
```



# CommandBars Property

Returns a [CommandBars](#) object that represents the Microsoft Excel command bars. Read-only.

## Remarks

Used with the **Application** object, this property returns the set of built-in and custom command bars available to the application.

When a workbook is embedded in another application and activated by the user by double-clicking the workbook, using this property with a **Workbook** object returns the set of Microsoft Excel command bars available within the other application. At all other times, using this property with a **Workbook** object returns **Nothing**.

There is no programmatic way to return the set of command bars attached to a workbook.

## Example

This example deletes all custom command bars that aren't visible.

```
For Each bar In Application.CommandBars  
    If Not bar.BuiltIn And Not bar.Visible Then bar.Delete  
Next
```



# CommandText Property

Returns or sets the command string for the specified data source. Read/write **Variant**.

## Remarks

You should use the **CommandText** property instead of the **SQL** property, which now exists primarily for compatibility with earlier versions of Microsoft Excel. If you use both properties, the **CommandText** property's value takes precedence.

For OLE DB sources, the [CommandType](#) property describes the value of the **CommandText** property.

For ODBC sources, the **CommandText** property functions exactly like the **SQL** property, and setting the property causes the data to be refreshed.

## Example

This example sets the command string for the first query table's ODBC data source. Note that the command string is an SQL statement.

```
Set qtQtrResults = _  
    Workbooks(1).Worksheets(1).QueryTables(1)  
With qtQtrResults  
    .CommandType = xlCmdSQL  
    .CommandText = _  
        "Select ProductID From Products Where ProductID < 10"  
    .Refresh  
End With
```



[Show All](#)

# CommandType Property

Returns or sets one of the **XlCmdType** constants listed in the following table. The constant that is returned or set describes the value of the [CommandText](#) property. The default value is **xlCmdSQL**. Read/write [XlCmdType](#).

XlCmdType can be one of these XlCmdType constants.

**xlCmdCube**. Contains a [cube](#) name for an [OLAP](#) data source.

**xlCmdDefault**. Contains command text that the OLE DB provider understands.

**xlCmdSql**. Contains an SQL statement.

**xlCmdTable**. Contains a table name for accessing OLE DB data sources.

*expression*.**CommandType**

## Remarks

You can set the **CommandType** property only if the value of the [QueryType](#) property for the query table or PivotTable cache is **xlOLEDBQuery**.

If the value of the **CommandType** property is **xlCmdCube**, you cannot change this value if there is a PivotTable report associated with the query table.

## Example

This example sets the command string for the first query table's ODBC data source. The command string is an SQL statement.

```
Set qtQtrResults = _  
    Workbooks(1).Worksheets(1).QueryTables(1)  
With qtQtrResults  
    .CommandType = xlCmdSQL  
    .CommandText = _  
        "Select ProductID From Products Where ProductID < 10"  
    .Refresh  
End With
```



[Show All](#)

# CommandUnderlines Property

Returns or sets the state of the command underlines in Microsoft Excel for the Macintosh. Can be one of the following [xlCommandUnderlines](#) constants. Read/write **Long**.

xlCommandUnderlines can be one of these xlCommandUnderlines constants.

**xlCommandUnderlinesOn**

**xlCommandUnderlinesOff**

**xlCommandUnderlinesAutomatic**

*expression*.**CommandUnderlines**

*expression* Required. An expression that returns one of the objects in the Applies To list.

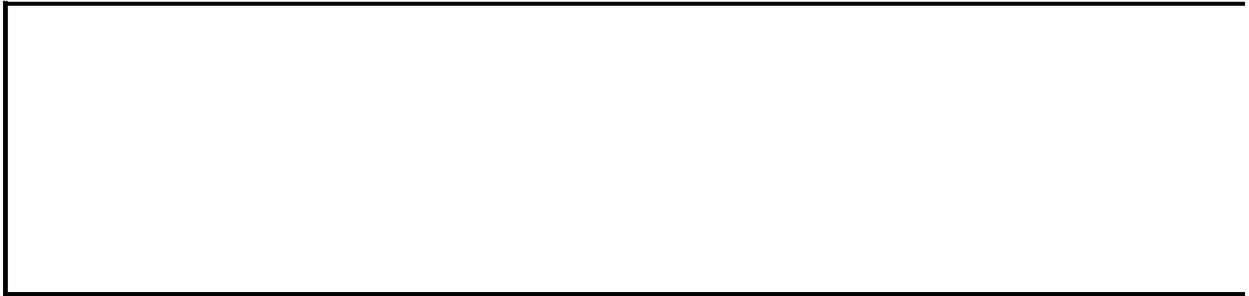
## Remarks

In Microsoft Excel for Windows, reading this property always returns **xlCommandUnderlinesOn**, and setting this property to anything other than **xlCommandUnderlinesOn** is an error.

## Example

This example turns off command underlines in Microsoft Excel for the Macintosh.

```
Application.CommandUnderlines = xlCommandUnderlinesOff
```



[Show All](#)

# Comment Property

 [Comment property as it applies to the \*\*Range\*\* object.](#)

Returns a **Comment** object that represents the comment associated with the cell in the upper-left corner of the range. Read-only **Comment** object.

*expression*.**Comment**

*expression* Required. An expression that returns a **Range** object.

 [Comment property as it applies to the \*\*Scenario\*\* object.](#)

Returns or sets the comment associated with the scenario. The comment text cannot exceed 255 characters. Read/write **String**.

*expression*.**Comment**

*expression* Required. An expression that returns a **Scenario** object.

## Example

This example sets the comment for scenario one on Sheet1.

```
worksheets("Sheet1").Scenarios(1).Comment = _  
    "Worst case July 1993 sales"
```



# Comments Property

Returns a [Comments](#) collection that represents all the comments for the specified worksheet. Read-only.

For information about returning a single member of a collection, see [Returning an Object from a Collection](#).

## Example

This example deletes all comments added by Jean Selva on the active sheet.

```
For Each c in ActiveSheet.Comments
    If c.Author = "Jean Selva" Then c.Delete
Next
```



[Show All](#)

# ConflictResolution Property

Returns or sets the way conflicts are to be resolved whenever a shared workbook is updated. Read/write [XISaveConflictResolution](#).

XISaveConflictResolution can be one of these XISaveConflictResolution constants.

**xlLocalSessionChanges.** The local user's changes are always accepted.

**xlOtherSessionChanges.** The local user's changes are always rejected.

**xlUserResolution.** A dialog box asks the user to resolve the conflict.

*expression*.**ConflictResolution**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example causes the local user's changes to be accepted whenever there's a conflict in the shared workbook.

```
ActiveWorkbook.ConflictResolution = xlLocalSessionChanges
```



[Show All](#)

# Connection Property

Returns or sets a string that contains one of the following: OLE DB settings that enable Microsoft Excel to connect to an OLE DB data source; ODBC settings that enable Microsoft Excel to connect to an ODBC data source; a URL that enables Microsoft Excel to connect to a Web data source; the path to and file name of a text file, or the path to and file name of a file that specifies a database or Web query. Read/write **Variant**.

## Remarks

Setting the **Connection** property doesn't immediately initiate the connection to the data source. You must use the [Refresh](#) method to make the connection and retrieve the data.

When using an [offline cube file](#), set the [UseLocalConnection](#) property to **True** and use the [LocalConnection](#) property instead of the **Connection** property.

For more information about the connection string syntax, see the [Add](#) method of the **QueryTables** collection and the [Add](#) method of the **PivotCaches** collection.

Alternatively, you may choose to access a data source directly by using the Microsoft ActiveX Data Objects (ADO) library instead.

## Example

This example creates a new PivotTable cache based on an [OLAP provider](#), and then it creates a new PivotTable report based on the cache, at cell A3 on the active worksheet.

```
With ActiveWorkbook.PivotCaches.Add(SourceType:=xlExternal)
    .Connection = _
        "OLEDB;Provider=MSOLAP;Location=srvdata;Initial Catalog=Nati
    .MaintainConnection = True
    .CreatePivotTable TableDestination:=Range("A3"), _
        TableName:= "PivotTable1"
End With
With ActiveSheet.PivotTables("PivotTable1")
    .SmallGrid = False
    .PivotCache.RefreshPeriod = 0
    With .CubeFields("[state]")
        .Orientation = xlColumnField
        .Position = 0
    End With
    With .CubeFields("[Measures].[Count Of au_id]")
        .Orientation = xlDataField
        .Position = 0
    End With
End With
```

This example supplies new ODBC connection information for the first query table on the first worksheet.

```
Worksheets(1).QueryTables(1) _
    .Connection := "ODBC;DSN=96SalesData;UID=Rep21;PWD=NUyHwYQI;"
```

This example specifies a text file.

```
Worksheets(1).QueryTables(1) _
    Connection := "TEXT;C:\My Documents\19980331.txt"
```



# ConnectionSiteCount Property

Returns the number of connection sites on the specified shape. Read-only **Long**.

## Example

This example adds two rectangles to myDocument and joins them with two connectors. The beginnings of both connectors attach to connection site one on the first rectangle; the ends of the connectors attach to the first and last connection sites of the second rectangle.

```
Set myDocument = Worksheets(1)
Set s = myDocument.Shapes
Set firstRect = s.AddShape(msoShapeRectangle, _
    100, 50, 200, 100)
Set secondRect = s.AddShape(msoShapeRectangle, _
    300, 300, 200, 100)
lastsite = secondRect.ConnectionSiteCount
With s.AddConnector(msoConnectorCurve, _
    0, 0, 100, 100).ConnectorFormat
    .BeginConnect ConnectedShape:=firstRect, _
        ConnectionSite:=1
    .EndConnect ConnectedShape:=secondRect, _
        ConnectionSite:=1
End With
With s.AddConnector(msoConnectorCurve, _
    0, 0, 100, 100).ConnectorFormat
    .BeginConnect ConnectedShape:=firstRect, _
        ConnectionSite:=1
    .EndConnect ConnectedShape:=secondRect, _
        ConnectionSite:=lastsite
End With
```



[Show All](#)

# Connector Property

**True** if the specified shape is a connector. Read-only [MsoTriState](#).

MsoTriState can be one of these MsoTriState constants.

**msoCTrue**

**msoFalse**

**msoTriStateMixed**

**msoTriStateToggle**

**msoTrue** The specified shape is a connector.

## Example

This example deletes all connectors on myDocument.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes
  For i = .Count To 1 Step -1
    With .Item(i)
      If .Connector Then .Delete
    End With
  Next
End With
```



# ConnectorFormat Property

Returns a [ConnectorFormat](#) object that contains connector formatting properties. Applies to **Shape** or **ShapeRange** objects that represent connectors. Read-only.

## Example

This example adds two rectangles to myDocument, attaches them with a connector, automatically reroutes the connector along the shortest path, and then detaches the connector from the rectangles.

```
Set myDocument = Worksheets(1)
Set s = myDocument.Shapes
Set firstRect = s.AddShape(msoShapeRectangle, 100, 50, 200, 100)
Set secondRect = s.AddShape(msoShapeRectangle, 300, 300, 200, 100)
Set c = s.AddConnector(msoConnectorCurve, 0, 0, 0, 0)
with c.ConnectorFormat
    .BeginConnect firstRect, 1
    .EndConnect secondRect, 1
    c.RerouteConnections
    .BeginDisconnect
    .EndDisconnect
End With
```



[Show All](#)

# ConsolidationFunction Property

Returns the function code used for the current consolidation. Can be one of the following [XlConsolidationFunction](#). Read-only **Long**.

XlConsolidationFunction can be one of these XlConsolidationFunction constants.

**xlAverage**

**xlCount**

**xlCountNums**

**xlMax**

**xlMin**

**xlProduct**

**xlStDev**

**xlStDevP**

**xlSum**

**xlUnknown**

**xlVar**

**xlVarP**

## Example

This example displays a message box if the current consolidation is using the SUM function.

```
If Worksheets("Sheet1").ConsolidationFunction = xlSum Then  
    MsgBox "Sheet1 uses the SUM function for consolidation."  
End If
```



# ConsolidationOptions Property

Returns a three-element array of consolidation options, as shown in the following table. If the element is **True**, that option is set. Read-only **Variant**.

<b>Element</b>	<b>Meaning</b>
1	Use labels in top row
2	Use labels in left column
3	Create links to source data

## Example

This example displays the consolidation options for Sheet1. The list appears on a new worksheet created by the example.

```
Set newSheet = Worksheets.Add
aOptions = Worksheets("Sheet1").ConsolidationOptions
newSheet.Range("A1").Value = "Use labels in top row"
newSheet.Range("A2").Value = "Use labels in left column"
newSheet.Range("A3").Value = "Create links to source data"
For i = 1 To 3
    If aOptions(i) = True Then
        newSheet.Cells(i, 2).Value = "True"
    Else
        newSheet.Cells(i, 2).Value = "False"
    End If
Next i
newSheet.Columns("A:B").AutoFit
```



# ConsolidationSources Property

Returns an array of string values that name the source sheets for the worksheet's current consolidation. Returns **Empty** if there's no consolidation on the sheet. Read-only **Variant**.

## Example

This example displays the names of the source ranges for the consolidation on Sheet1. The list appears on a new worksheet created by the example.

```
Set newSheet = Worksheets.Add
newSheet.Range("A1").Value = "Consolidation Sources"
aSources = Worksheets("Sheet1").ConsolidationSources
If IsEmpty(aSources) Then
    newSheet.Range("A2").Value = "none"
Else
    For i = 1 To UBound(aSources)
        newSheet.Cells(i + 1, 1).Value = aSources(i)
    Next i
End If
newSheet.Columns("A:B").AutoFit
```



# ConstrainNumeric Property

**True** if handwriting recognition is limited to numbers and punctuation only.  
Read/write **Boolean**.

**Note** This property is available only if you're using Microsoft Windows for Pen Computing. If you try to set this property under any other operating system, an error occurs.

## Example

This example limits handwriting recognition to numbers and punctuation only if Microsoft Windows for Pen Computing is running.

```
If Application.WindowsForPens Then  
    Application.ConstrainNumeric = True  
End If
```



# Container Property

Returns the object that represents the container application for the specified OLE object. Read-only **Object**.

## Remarks

This property provides a way to access the object model of the container application if an Excel workbook is opened within a host application such as Microsoft Internet Explorer.

--

# Contrast Property

Returns or sets the contrast for the specified picture or OLE object. The value for this property must be a number from 0.0 (the least contrast) to 1.0 (the greatest contrast). Read/write **Single**.

*expression*.**Contrast**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example sets the contrast for shape one on myDocument. Shape one must be either a picture or an OLE object.

```
Set myDocument = Worksheets(1)  
myDocument.Shapes(1).PictureFormat.Contrast = 0.8
```



# ControlCharacters Property

**True** if Microsoft Excel displays control characters for right-to-left languages.  
Read/write **Boolean**.

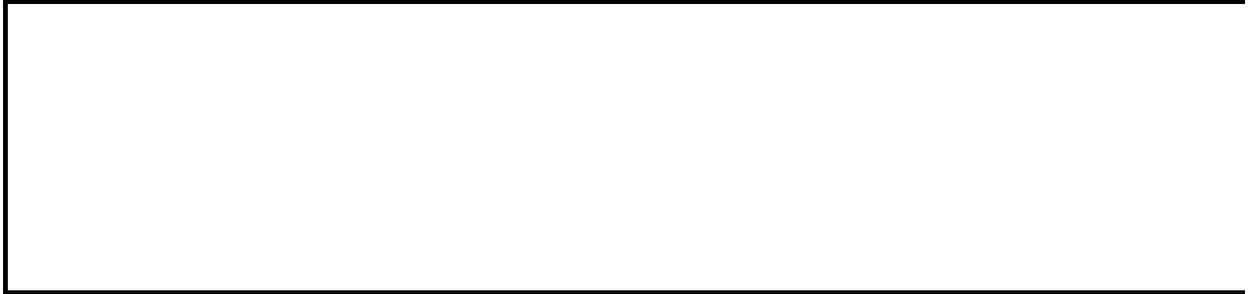
## **Remarks**

This property can be set only when right to left language support has been installed and selected.

## Example

This example sets Microsoft Excel to interpret control characters.

```
Application.ControlCharacters = True
```



[Show All](#)

# ControlFormat Property

Returns a [ControlFormat](#) object that contains [Microsoft Excel control](#) properties. Read-only.

## Example

This example removes the selected item from a list box. If Shapes(2) doesn't represent a list box, this example fails.

```
Set lbcf = Worksheets(1).Shapes(2).ControlFormat  
lbcf.RemoveItem lbcf.ListIndex
```



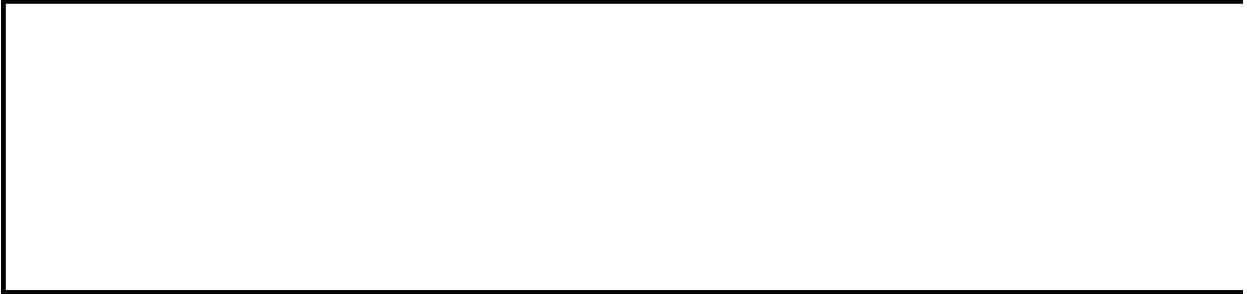
# CopyObjectsWithCells Property

**True** if objects are cut, copied, extracted, and sorted with cells. Read/write **Boolean**.

## Example

This example sets Microsoft Excel to cut, copy, extract, and sort objects with cells.

```
Application.CopyObjectsWithCells = True
```



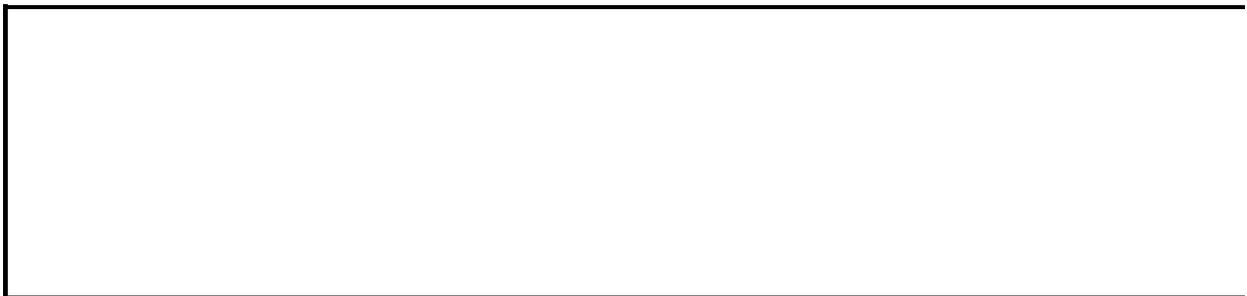
# Corners Property

Returns a [Corners](#) object that represents the corners of a 3-D chart. Read-only.

## Example

This example selects the corners of Chart1. The example should be run on a 3-D chart (the **Select** method fails on any other chart type).

```
With Charts("Chart1")  
    .Activate  
    .Corners.Select  
End With
```



# CorrectCapsLock Property

**True** if Microsoft Excel automatically corrects accidental use of the CAPS LOCK key. Read/write **Boolean**.

## Example

This example enables Microsoft Excel to automatically correct accidental use of the CAPS LOCK key.

```
Application.AutoCorrect.CorrectCapsLock = True
```



# CorrectSentenceCap Property

**True** if Microsoft Excel automatically corrects sentence (first word) capitalization. Read/write **Boolean**.

## Example

This example enables Microsoft Excel to automatically correct sentence capitalization.

```
Application.AutoCorrect.CorrectSentenceCap = True
```



[Show All](#)

# Count Property

[Count property as it applies to the \*\*Adjustments\*\*, \*\*CanvasShapes\*\*, \*\*DiagramNodeChildren\*\*, \*\*DiagramNodes\*\*, \*\*ListObjects\*\*, \*\*ListRows\*\*, \*\*ListColumns\*\*, and \*\*ShapeNodes\*\* objects.](#)

Returns the number of objects in the collection. Read-only **Integer**.

*expression*.**Count**

*expression* Required. An expression that returns one of the above objects.

[Count property as it applies to all other objects in the \*\*Applies To\*\* list.](#)

Returns the number of objects in the collection. Read-only **Long**.

*expression*.**Count**

*expression* Required. An expression that returns one of the objects in the **Applies To** list.

## Example

This example displays the number of columns in the selection on Sheet1. The code also tests for a multiple-area selection; if one exists, the code loops on the areas of the multiple-area selection.

```
Sub DisplayColumnCount()  
    Dim iAreaCount As Integer  
    Dim i As Integer  
  
    Worksheets("Sheet1").Activate  
    iAreaCount = Selection.Areas.Count  
  
    If iAreaCount <= 1 Then  
        MsgBox "The selection contains " & Selection.Columns.Count &  
    Else  
        For i = 1 To iAreaCount  
            MsgBox "Area " & i & " of the selection contains " & _  
                Selection.Areas(i).Columns.Count & " columns."  
        Next i  
    End If  
End Sub
```

This example makes the last character in cell A1 a superscript character.

```
Sub MakeSuperscript()  
    Dim n As Integer  
  
    n = Worksheets("Sheet1").Range("A1").Characters.Count  
    Worksheets("Sheet1").Range("A1").Characters(n, 1) _  
        .Font.Superscript = True  
End Sub
```



# CreateBackup Property

**True** if a backup file is created when this file is saved. Read-only **Boolean**.

## Example

This example displays a message if a backup file is created when the active workbook is saved.

```
If ActiveWorkbook.CreateBackup = True Then  
    MsgBox "Remember, there is a backup copy of this workbook"  
End If
```



[Show All](#)

# Creator Property

 [Creator property as it applies to the \*\*Adjustments\*\*, \*\*CalloutFormat\*\*, \*\*ColorFormat\*\*, \*\*DiagramNode\*\*, \*\*DiagramNodeChildren\*\*, \*\*DiagramNodes\*\*, \*\*FillFormat\*\*, \*\*ListColumn\*\*, \*\*ListDataFormat\*\*, \*\*LineFormat\*\*, \*\*ListObject\*\*, \*\*ListObjects\*\*, \*\*ListRow\*\*, \*\*ListRows\*\*, \*\*PictureFormat\*\*, \*\*ShadowFormat\*\*, \*\*ShapeNode\*\*, \*\*ShapeNodes\*\*, \*\*TextEffectFormat\*\*, and \*\*ThreeDFormat\*\* objects.](#)

Returns a 32-bit integer that indicates the application in which this object was created. If the object was created in Microsoft Excel, this property returns the string XCEL, which is equivalent to the hexadecimal number 5843454C. Read-only **Long**.

*expression*.**Creator**

*expression* Required. An expression that returns one of the above objects.

 [Creator property as it applies to all other objects in the Applies To list.](#)

Returns a 32-bit integer that indicates the application in which this object was created. If the object was created in Microsoft Excel, this property returns the string XCEL, which is equivalent to the hexadecimal number 5843454C. Read-only **xlCreatorCode**.

*expression*.**Creator**

*expression* Required. An expression that returns all other objects in the Applies To list.

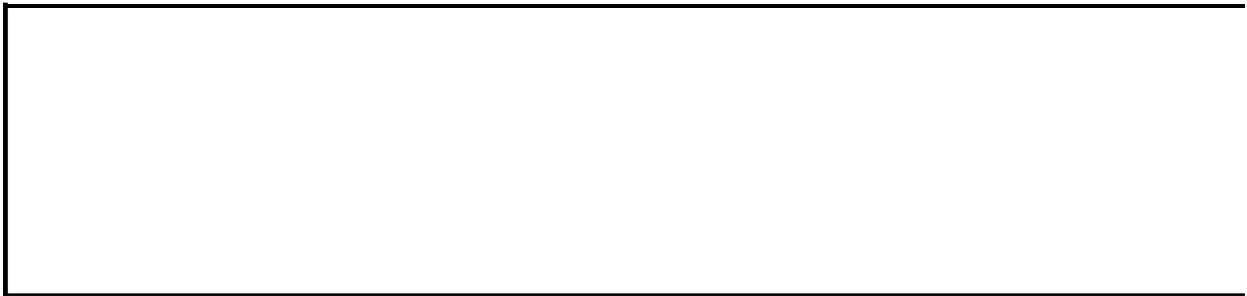
## Remarks

The **Creator** property is designed to be used in Microsoft Excel for the Macintosh, where each application has a four-character creator code. For example, Microsoft Excel has the creator code XCEL.

## Example

This example displays a message about the creator of an Excel workbook.

```
Sub FindCreator()  
  
    Dim myObject As Excel.Workbook  
    Set myObject = ActiveWorkbook  
    If myObject.Creator = &h5843454c Then  
        MsgBox "This is a Microsoft Excel object."  
    Else  
        MsgBox "This is not a Microsoft Excel object."  
    End If  
  
End Sub
```



# Criteria1 Property

Returns the first filtered value for the specified column in a filtered range. Read-only **Variant**.

## Example

The following example sets a variable to the value of the **Criteria1** property of the filter for the first column in the filtered range on the Crew worksheet.

```
With Worksheets("Crew")
    If .AutoFilterMode Then
        With .AutoFilter.Filters(1)
            If .On Then c1 = .Criteria1
        End With
    End If
End With
```



# Criteria2 Property

Returns the second filtered value for the specified column in a filtered range.  
Read-only **Variant**.

## Remarks

If you try to access the **Criteria2** property for a filter that does not use two criteria, an error will occur. Check that the **Operator** property of a **Filter** object doesn't equal zero (0) before trying to access the **Criteria2** property.

## Example

The following example sets a variable to the value of the **Criteria2** property of the filter for the first column in the filtered range on the Crew worksheet.

```
With Worksheets("Crew")
    If .AutoFilterMode Then
        With .AutoFilter.Filters(1)
            If .On And .Operator Then
                c2 = .Criteria2
            Else
                c2 = "Not set"
            End If
        End With
    End If
End With
```



# CropBottom Property

Returns or sets the number of points that are cropped off the bottom of the specified picture or OLE object. Read/write **Single**.

*expression*.**CropBottom**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Cropping is calculated relative to the original size of the picture. For example, if you insert a picture that is originally 100 points high, rescale it so that it's 200 points high, and then set the **CropBottom** property to 50, 100 points (not 50) will be cropped off the bottom of your picture.

## Example

This example crops 20 points off the bottom of shape three on myDocument. For the example to work, shape three must be either a picture or an OLE object.

```
Set myDocument = Worksheets(1)
myDocument.Shapes(3).PictureFormat.CropBottom = 20
```

Using this example, you can specify the percentage you want to crop off the bottom of the selected shape, regardless of whether the shape has been scaled. For the example to work, the selected shape must be either a picture or an OLE object.

```
percentToCrop = InputBox( _
    "What percentage do you want to crop off" & _
    " the bottom of this picture?")
Set shapeToCrop = ActiveWindow.Selection.ShapeRange(1)
With shapeToCrop.Duplicate
    .ScaleHeight 1, True
    origHeight = .Height
    .Delete
End With
cropPoints = origHeight * percentToCrop / 100
shapeToCrop.PictureFormat.CropBottom = cropPoints
```



# CropLeft Property

Returns or sets the number of points that are cropped off the left side of the specified picture or OLE object. Read/write **Single**.

*expression*.**CropLeft**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Cropping is calculated relative to the original size of the picture. For example, if you insert a picture that is originally 100 points wide, rescale it so that it's 200 points wide, and then set the **CropLeft** property to 50, 100 points (not 50) will be cropped off the left side of your picture.

## Example

This example crops 20 points off the left side of shape three on myDocument. For the example to work, shape three must be either a picture or an OLE object.

```
Set myDocument = Worksheets(1)
myDocument.Shapes(3).PictureFormat.CropLeft = 20
```

Using this example, you can specify the percentage you want to crop off the left side of the selected shape, regardless of whether the shape has been scaled. For the example to work, the selected shape must be either a picture or an OLE object.

```
percentToCrop = InputBox( _
    "What percentage do you want to crop" & _
    " off the left of this picture?")
Set shapeToCrop = ActiveWindow.Selection.ShapeRange(1)
With shapeToCrop.Duplicate
    .ScaleWidth 1, True
    origWidth = .Width
    .Delete
End With
cropPoints = origWidth * percentToCrop / 100
shapeToCrop.PictureFormat.CropLeft = cropPoints
```



# CropRight Property

Returns or sets the number of points that are cropped off the right side of the specified picture or OLE object. Read/write **Single**.

*expression*.**CropRight**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Cropping is calculated relative to the original size of the picture. For example, if you insert a picture that is originally 100 points wide, rescale it so that it's 200 points wide, and then set the **CropRight** property to 50, 100 points (not 50) will be cropped off the right side of your picture.

## Example

This example crops 20 points off the right side of shape three on myDocument. For this example to work, shape three must be either a picture or an OLE object.

```
Set myDocument = Worksheets(1)
myDocument.Shapes(3).PictureFormat.CropRight = 20
```

Using this example, you can specify the percentage you want to crop off the right side of the selected shape, regardless of whether the shape has been scaled. For the example to work, the selected shape must be either a picture or an OLE object.

```
percentToCrop = InputBox( _
    "What percentage do you want to crop" & _
    " off the right of this picture?")
Set shapeToCrop = ActiveWindow.Selection.ShapeRange(1)
With shapeToCrop.Duplicate
    .ScaleWidth 1, True
    origWidth = .Width
    .Delete
End With
cropPoints = origWidth * percentToCrop / 100
shapeToCrop.PictureFormat.CropRight = cropPoints
```



# CropTop Property

Returns or sets the number of points that are cropped off the top of the specified picture or OLE object. Read/write **Single**.

*expression*.**CropTop**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Cropping is calculated relative to the original size of the picture. For example, if you insert a picture that is originally 100 points high, rescale it so that it's 200 points high, and then set the **CropTop** property to 50, 100 points (not 50) will be cropped off the top of your picture.

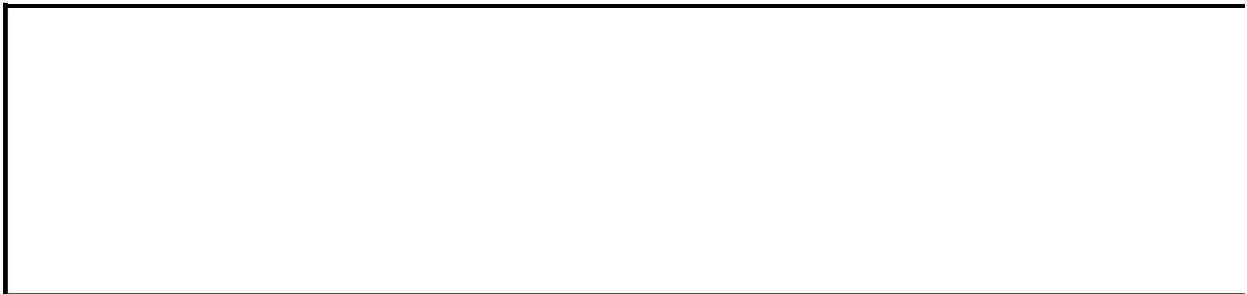
## Example

This example crops 20 points off the top of shape three on myDocument. For the example to work, shape three must be either a picture or an OLE object.

```
Set myDocument = Worksheets(1)
myDocument.Shapes(3).PictureFormat.CropTop = 20
```

This example allows you to specify the percentage you want to crop off the top of the selected shape, regardless of whether the shape has been scaled. For the example to work, the selected shape must be either a picture or an OLE object.

```
percentToCrop = InputBox( _
    "What percentage do you want to crop" & _
    " off the top of this picture?")
Set shapeToCrop = ActiveWindow.Selection.ShapeRange(1)
With shapeToCrop.Duplicate
    .ScaleHeight 1, True
    origHeight = .Height
    .Delete
End With
cropPoints = origHeight * percentToCrop / 100
shapeToCrop.PictureFormat.CropTop = cropPoints
```



# Crosses Property

Returns or sets the point on the specified axis where the other axis crosses.  
Read/write **Long**.

Can be one of the **XlAxisCrosses** constants listed in the following table.

<b>Constant</b>	<b>Meaning</b>
<b>xlAxisCrossesAutomatic</b>	Microsoft Excel sets the axis crossing point.
<b>xlMinimum</b>	The axis crosses at the minimum value.
<b>xlMaximum</b>	The axis crosses at the maximum value.
<b>xlAxisCrossesCustom</b>	The <a href="#">CrossesAt</a> property specifies the axis crossing point.

## Remarks

This property isn't available for radar charts. For 3-D charts, this property indicates where the plane defined by the category axes crosses the value axis.

This property can be used for both category and value axes. On the category axis, **xlMinimum** sets the value axis to cross at the first category, and **xlMaximum** sets the value axis to cross at the last category.

Note that **xlMinimum** and **xlMaximum** can have different meanings, depending on the axis.

## Example

This example sets the value axis in Chart1 to cross the category axis at the maximum x value.

```
Charts("Chart1").Axes(xlCategory).Crosses = xlMaximum
```



# CrossesAt Property

Returns or sets the point on the value axis where the category axis crosses it. Applies only to the value axis. Read/write **Double**.

*expression*.**CrossesAt**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

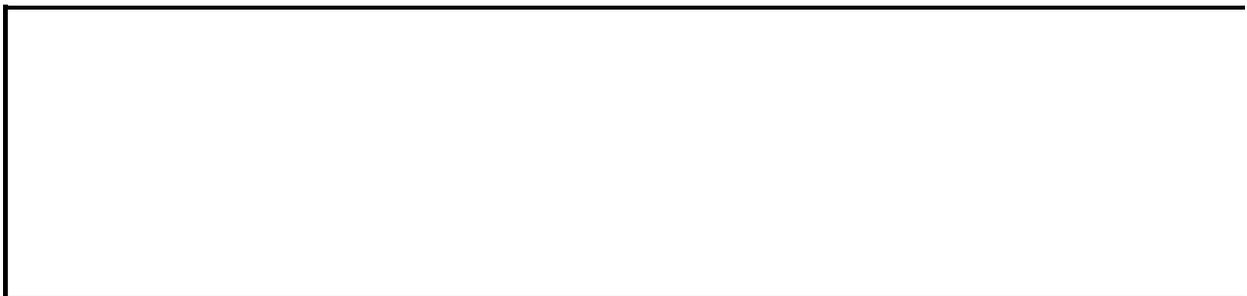
Setting this property causes the [Crosses](#) property to change to **xlAxisCrossesCustom**.

This property cannot be used on radar charts. For 3-D charts, this property indicates where the plane defined by the category axes crosses the value axis.

## Example

This example sets the category axis in the active chart to cross the value axis at value 3.

```
Sub Chart()  
    ' Create a sample source of data.  
    Range("A1") = "2"  
    Range("A2") = "4"  
    Range("A3") = "6"  
    Range("A4") = "3"  
  
    ' Create a chart based on the sample source of data.  
    Charts.Add  
  
    With ActiveChart  
        .ChartType = xlLineMarkersStacked  
        .SetSourceData Source:=Sheets("Sheet1").Range("A1:A4"), Plot  
        .Location Where:=xlLocationAsObject, Name:="Sheet1"  
    End With  
  
    ' Set the category axis to cross the value axis at value 3.  
    ActiveChart.Axes(xlValue).Select  
    Selection.CrossesAt = 3  
  
End Sub
```



[Show All](#)

# CubeField Property

Returns the [CubeField](#) object from which the specified PivotTable field is descended. Read-only.

*expression*.**CubeField**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example creates a list of the cube field names for all the hierarchy fields in the first [Online Analytical Processing \(OLAP\)](#) -based PivotTable report on the first worksheet. This example assumes a PivotTable report exists in the first worksheet.

```
Sub UseCubeField()  
  
    Dim objNewSheet As Worksheet  
    Set objNewSheet = Worksheets.Add  
    objNewSheet.Activate  
    intRow = 1  
  
    For Each objPF in _  
        Worksheets(1).PivotTables(1).PivotFields  
        If objPF.CubeField.CubeFieldType = xlHierarchy Then  
            objNewSheet.Cells(intRow, 1).Value = objPF.Name  
            intRow = intRow + 1  
        End If  
    Next objPF  
  
End Sub
```



[Show All](#)

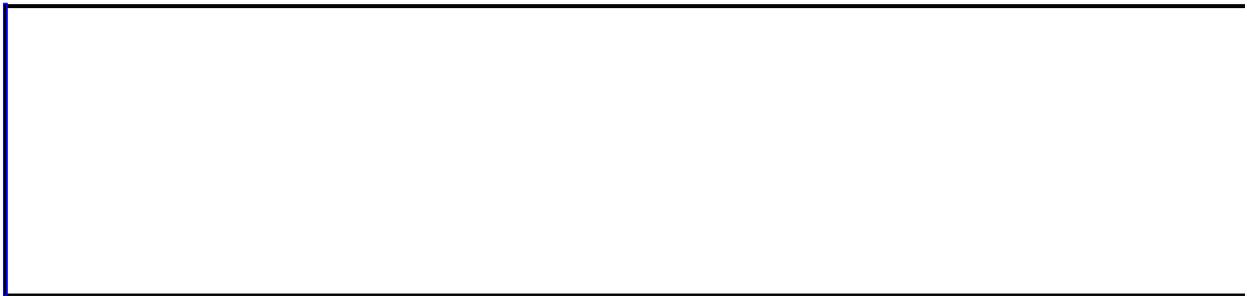
# CubeFields Property

Returns the [CubeFields](#) collection. Each [CubeField](#) object contains the properties of the [cube](#) field element. Read-only.

## Example

This example creates a list of [cube](#) field names for the data fields in the first [OLAP](#)-based PivotTable report on Sheet1.

```
Set objNewSheet = Worksheets.Add  
objNewSheet.Activate  
intRow = 1  
For Each objCubeFld In Worksheets("Sheet1").PivotTables(1).CubeFields  
    If objCubeFld.Orientation = xlDataField Then  
        objNewSheet.Cells(intRow, 1).Value = objCubeFld.Name  
        intRow = intRow + 1  
    End If  
Next objCubeFld
```



[Show All](#)

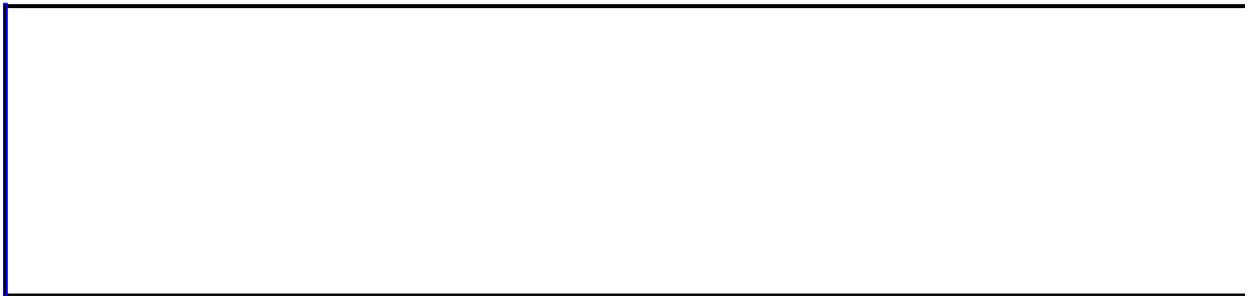
# CubeFieldType Property

Indicates whether the [OLAP cube](#) field is a hierarchy field or a measure field. Can be one of the following **XICubeFieldType** constants: **xlHierarchy** or **xlMeasure**. Read-only **XICubeFieldType**.

## Example

This example creates a list of [cube](#) field names for the measure fields in the first [OLAP](#)-based PivotTable report on Sheet1.

```
Set objNewSheet = Worksheets.Add  
objNewSheet.Activate  
intRow = 1  
For Each objCubeFld in Worksheets("Sheet1").PivotTables(1).CubeField  
    If objCubeFld.CubeFieldType = xlMeasure Then  
        objNewSheet.Cells(intRow, 1).Value = objCubeFld.Name  
        intRow = intRow + 1  
    End If  
Next objCubeFld
```



# CurrentArray Property

If the specified cell is part of an array, returns a [Range](#) object that represents the entire array. Read-only.

## Example

This example assumes that cell A1 on Sheet1 is the active cell and that the active cell is part of an array that includes cells A1:A10. The example selects cells A1:A10 on Sheet1.

```
ActiveCell.CurrentArray.Select
```



# CurrentPage Property

Returns or sets the current page showing for the page field (valid only for page fields). Read/write **PivotItem**.

*expression*.**CurrentPage**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example returns the current page name for the PivotTable report on Sheet1 in the string variable strPgName.

```
Set pvtTable = Worksheets("Sheet1").Range("A3").PivotTable  
strPgName = pvtTable.PivotFields("Country").CurrentPage.Name
```



# CurrentPageList Property

Returns or sets an array of strings corresponding to the list of items included in a multiple-item page field of a PivotTable report. Read/write **Variant**.

*expression*.**CurrentPageList**

*expression* Required. An expression that returns a [PivotField](#) object.

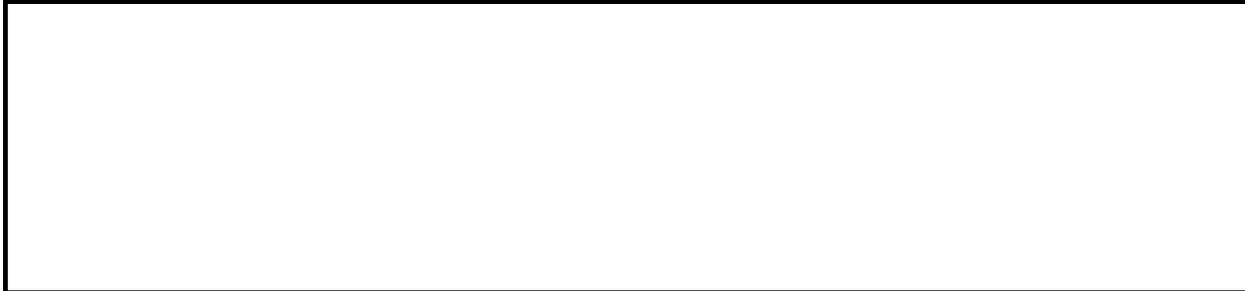
## Remarks

To avoid run-time errors, the data source must be an OLAP source, the field chosen must currently be in the Page position, and the [EnableMultiplePageItems](#) property must be set to **True**.

## Example

This example sets the page field to list the "Food" items of the PivotTable report. It assumes a PivotTable exists on the active worksheet.

```
Sub UseCurrentPageList()  
  
    Dim pvtTable As PivotTable  
    Dim pvtField As PivotField  
  
    Set pvtTable = ActiveSheet.PivotTables(1)  
    Set pvtField = pvtTable.PivotFields("[Product]")  
  
    ' To avoid run-time errors set the following property to True.  
    pvtTable.CubeFields("[Product]").EnableMultiplePageItems = True  
  
    ' Set the page list to "Food".  
    pvtField.CurrentPageList = "[Product].[All Products].[Food]"  
  
End Sub
```



# CurrentPageName Property

Returns or sets the currently displayed page of the specified PivotTable report. The name of the page appears in the page field. Note that this property works only if the currently displayed page already exists. Read/write **String**.

## **Remarks**

This property applies to PivotTables that are connected to an OLAP data source. Attempting to return or set this property with a PivotTable that is not connected to an OLAP data source will result in a run-time error.

## Example

This example sets the name of the currently displayed page of the first PivotTable report on the active worksheet to "USA."

```
ActiveSheet.PivotTables("PivotTable1") _  
    .PivotFields("[Customers]").CurrentPageName = _  
    "[Customers].[All Customers].[USA]"
```



# CurrentRegion Property

Returns a [Range](#) object that represents the current region. The current region is a range bounded by any combination of blank rows and blank columns. Read-only.

## Remarks

This property is useful for many operations that automatically expand the selection to include the entire current region, such as the [AutoFormat](#) method.

This property cannot be used on a protected worksheet.

## Example

This example selects the current region on Sheet1.

```
Worksheets("Sheet1").Activate  
ActiveCell.CurrentRegion.Select
```

This example assumes that you have a table on Sheet1 that has a header row. The example selects the table, without selecting the header row. The active cell must be somewhere in the table before you run the example.

```
Set tbl = ActiveCell.CurrentRegion  
tbl.Offset(1, 0).Resize(tbl.Rows.Count - 1, _  
tbl.Columns.Count).Select
```



[Show All](#)

# Cursor Property

Returns or sets the appearance of the mouse pointer in Microsoft Excel.  
Read/write [XlMousePointer](#).

XlMousePointer can be one of these XlMousePointer constants.

**xlDefault** The default pointer.

**xlIBeam** The I-beam pointer.

**xlNorthwestArrow** The northwest-arrow pointer.

**xlWait** The hourglass pointer.

*expression*.**Cursor**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

The **Cursor** property isn't reset automatically when the macro stops running. You should reset the pointer to **xlDefault** before your macro stops running.

## Example

This example changes the mouse pointer to an I-beam, pauses, and then changes it to the default pointer.

```
Sub ChangeCursor()  
    Application.Cursor = xlIBeam  
    For x = 1 To 1000  
        For y = 1 to 1000  
            Next y  
        Next x  
    Application.Cursor = xlDefault  
End Sub
```



# CursorMovement Property

Returns or sets a value that indicates whether a visual cursor or a logical cursor is used. Can be one of the following constants: **xlVisualCursor** or **xlLogicalCursor**. Read/write **Long**.

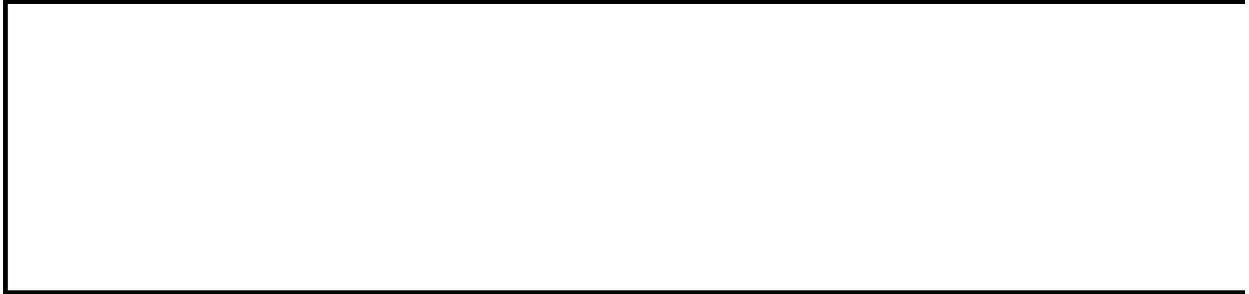
## Remarks

These constants may not be available to you, depending on the language support (U.S. English, for example) that you've selected or installed.

## Example

This example sets Microsoft Excel to use the visual cursor.

```
Application.CursorMovement = xlVisualCursor
```



# CustomDocumentProperties Property

Returns or sets a [DocumentProperties](#) collection that represents all the custom document properties for the specified workbook.

*expression*.**CustomDocumentProperties**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

This property returns the entire collection of custom document properties. Use the **Item** method to return a single member of the collection (a **DocumentProperty** object) by specifying either the name of the property or the collection index (as a number).

Because the **Item** method is the default method for the **DocumentProperties** collection, the following statements are identical:

```
CustomDocumentProperties.Item("Complete")  
CustomDocumentProperties("Complete")
```

Use the [BuiltinDocumentProperties](#) property to return the collection of built-in document properties.

## Example

This example displays the names and values of the custom document properties as a list on worksheet one.

```
rw = 1
Worksheets(1).Activate
For Each p In ActiveWorkbook.CustomDocumentProperties
    Cells(rw, 1).Value = p.Name
    Cells(rw, 2).Value = p.Value
    rw = rw + 1
Next
```



# CustomListCount Property

Returns the number of defined custom lists (including built-in lists). Read-only  
**Long.**

## Example

This example displays the number of custom lists that are currently defined.

```
MsgBox "There are currently " & Application.CustomListCount & _  
      " defined custom lists."
```



# CustomProperties Property

Returns a [CustomProperties](#) object representing the identifier information associated with a worksheet.

*expression*.**CustomProperties**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

For the **CustomProperties** property, identifier information for a worksheet can represent metadata for XML.

## Example

In this example, Microsoft Excel adds identifier information to the active worksheet and returns the name and value to the user.

```
Sub CheckCustomProperties()  
    Dim wksSheet1 As Worksheet  
  
    Set wksSheet1 = Application.ActiveSheet  
  
    ' Add metadata to worksheet.  
    wksSheet1.CustomProperties.Add _  
        Name:="Market", Value:="Nasdaq"  
  
    ' Display metadata.  
    With wksSheet1.CustomProperties.Item(1)  
        MsgBox .Name & vbTab & .Value  
    End With  
  
End Sub
```



[Show All](#)

# CustomSubtotalFunction Property

Returns the custom subtotal function field setting of a **PivotCell** object. Read-only [XLConsolidationFunction](#).

XLConsolidationFunction can be one of these XLConsolidationFunction constants.

**xlAverage**

**xlCount**

**xlCountNums**

**xlMax**

**xlMin**

**xlProduct**

**xlStDev**

**xlStDevP**

**xlSum**

**xlUnknown**

**xlVar**

**xlVarP**

*expression*.**CustomSubtotalFunction**

*expression* Required. An expression that returns a **PivotCell** object.

## Remarks

The **CustomSubtotalFunction** property will return an error if the **PivotCell** object type is not a custom subtotal. This property applies only to [non-OLAP source data](#).

## Example

This example determines if cell C20 contains a custom subtotal function that uses a consolidation function of count and then it notifies the user. The example assumes a PivotTable exists on the active worksheet.

```
Sub UseCustomSubtotalFunction()  
    On Error GoTo Not_A_Function  
  
    ' Determine if custom subtotal function is a count function.  
    If Application.Range("C20").PivotCell.CustomSubtotalFunction = x  
        MsgBox "The custom subtotal function is a Count."  
    Else  
        MsgBox "The custom subtotal function is not a Count."  
    End If  
    Exit Sub  
  
Not_A_Function:  
    MsgBox "The selected cell is not a custom subtotal function."  
  
End Sub
```



# CustomViews Property

Returns a [CustomViews](#) collection that represents all the custom views for the workbook.

For more information about returning a single object from a collection, see [Returning an Object from a Collection](#).

## Example

This example creates a new custom view named "Summary" in the active workbook.

```
ActiveWorkbook.CustomViews.Add "Summary", True, True
```



# CutCopyMode Property

Returns or sets the status of Cut or Copy mode. Can be **True**, **False**, or an **XLCutCopyMode** constant, as shown in the following tables. Read/write **Long**.

<b>Return value</b>	<b>Description</b>
<b>False</b>	Not in Cut or Copy mode
<b>xlCopy</b>	In Copy mode
<b>xlCut</b>	In Cut mode

<b>Set value</b>	<b>Description</b>
<b>False</b>	Cancels Cut or Copy mode and removes the moving border.
<b>True</b>	Cancels Cut or Copy mode and removes the moving border.

## Example

This example uses a message box to display the status of Cut or Copy mode.

```
Select Case Application.CutCopyMode
  Case Is = False
    MsgBox "Not in Cut or Copy mode"
  Case Is = xlCopy
    MsgBox "In Copy mode"
  Case Is = xlCut
    MsgBox "In Cut mode"
End Select
```



[Show All](#)

# DashStyle Property

Returns or sets the dash style for the specified line. Can be one of the [MsoLineDashStyle](#) constants. Read/write **Long**.

MsoLineDashStyle can be one of these MsoLineDashStyle constants.

**msoLineDash**

**msoLineDashDot**

**msoLineDashDotDot**

**msoLineDashStyleMixed**

**msoLineDashLongDash**

**msoLineDashLongDashDot**

**msoLineRoundDot**

**msoLineSolid**

**msoLineSquareDot**

*expression*.**DashStyle**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example adds a blue dashed line to myDocument.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddLine(10, 10, 250, 250).Line
    .DashStyle = msoLineDashDotDot
    .ForeColor.RGB = RGB(50, 0, 128)
End With
```



[Show All](#)

# DatabaseSort Property

When set to **True**, manual repositioning of items in a PivotTable field is allowed. Returns **True**, if the field has no manually positioned items. Read/write **Boolean**.

*expression.DatabaseSort*

*expression* Required. An expression that returns a [PivotField](#) object.

## Remarks

The **DatabaseSort** property returns **False** if the data source is not an [Online Analytical Processing \(OLAP\)](#) data source.

This property returns **True** if the data source is OLAP and neither custom ordering nor automatic sorting has been applied to the field.

Setting the **DatabaseSort** property to **True**, for an OLAP PivotTable, will remove any custom ordering or automatic sort applied to the field (in other words, the PivotTable reverts to the default behavior when the connection was made).

Setting the **DatabaseSort** property to **False** will cause the sort order to be the current order of the items, if no automatic sort is applied.

Setting the **DatabaseSort** property to either **True** or **False** causes an Update.

Setting the **DatabaseSort** property to **True** for a non-OLAP source or an OLAP data field causes a run-time error.

## Example

The following example determines if the data source is an OLAP data source and notifies the user. This example assumes an OLAP PivotTable exists on the active worksheet.

```
Sub UseDatabaseSort()  
  
    Dim pvtTable As PivotTable  
    Dim pvtField As PivotField  
  
    Set pvtTable = ActiveSheet.PivotTables(1)  
    Set pvtField = pvtTable.PivotFields("[Product].[Product Family]")  
  
    ' Determine source type for the PivotTable report.  
    If pvtField.DatabaseSort = True Then  
        MsgBox "The source is OLAP; you can manually reorder items."  
    Else  
        MsgBox "The data source might not be OLAP."  
    End If  
  
End Sub
```



# DataBinding Property

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Returns an [XmlDataBinding](#) object that represents the binding associated with the specified schema map. Read-only.

*expression*.**DataBinding**

*expression* Required. An expression that returns an [XmlMap](#) object.



# DataBodyRange Property

Returns a [Range](#) object that represents the range that contains the data area in the list between the header row and the insert row. Read-only.

## Example

This example selects the active data range in the list.

```
Worksheets("Sheet1").Activate  
ActiveWorksheet.ListObjects.Item(1).DataBodyRange.Select
```



# DataEntryMode Property

Returns or sets Data Entry mode, as shown in the following table. When in Data Entry mode, you can enter data only in the unlocked cells in the currently selected range. Read/write **Long**.

<b>Value</b>	<b>Meaning</b>
<b>xlOn</b>	Data Entry mode is turned on.
<b>xlOff</b>	Data Entry mode is turned off.
<b>xlStrict</b>	Data Entry mode is turned on, and pressing ESC won't turn it off.

## Example

This example turns off Data Entry mode if it's on.

```
If (Application.DataEntryMode = xlOn) Or _  
    (Application.DataEntryMode = xlStrict) Then  
    Application.DataEntryMode = xlOff  
End If
```



# DataField Property

Returns a [PivotField](#) object that corresponds to the selected data field.

*expression*.**DataField**

*expression* Required. An expression that returns a [PivotCell](#) object.

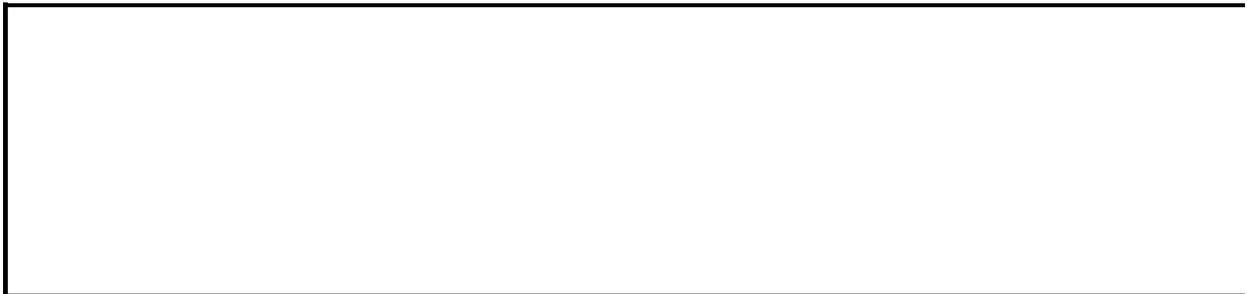
## Remarks

This property will return an error if the **PivotCell** object is not one of the allowed types: **XIPivotCellTypeValue**, **XIPivotCellTypeDataField**, **XIPivotCellTypeSubtotal**, **XIPivotCellTypeGrandTotal**.

## Example

This example determines if cell L10 is in the data field of the PivotTable and either returns the PivotTable field that corresponds to the cell by notifying the user, or handles the run-time error. The example assumes a PivotTable exists in the active worksheet.

```
Sub CheckDataField()  
    On Error GoTo Not_In_DataField  
    MsgBox Application.Range("L10").PivotCell.DataField  
    Exit Sub  
Not_In_DataField:  
    MsgBox "The selected range is not in the data field of the Pivot  
End Sub
```



# DataFields Property

Returns an object that represents either a single PivotTable field (a [PivotField](#) object) or a collection of all the fields (a [PivotFields](#) object) that are currently shown as data fields. Read-only.

*expression*.DataFields(*Index*)

*expression* Required. An expression that returns a **PivotTable** object.

**Index** Optional **Variant**. The field name or number (can be an array to specify more than one field).

## Example

This example adds the names for the PivotTable data fields to a list on a new worksheet.

```
Set nwSheet = Worksheets.Add  
nwSheet.Activate  
Set pvtTable = Worksheets("Sheet2").Range("A1").PivotTable  
rw = 0  
For Each pvtField In pvtTable.DataFields  
    rw = rw + 1  
    nwSheet.Cells(rw, 1).Value = pvtField.Name  
Next pvtField
```



# DataLabel Property

Returns a [DataLabel](#) object that represents the data label associated with the point or trendline. Read-only.

## Example

This example turns on the data label for point seven in series three in Chart1, and then it sets the data label color to blue.

```
With Charts("Chart1").SeriesCollection(3).Points(7)
    .HasDataLabel = True
    .ApplyDataLabels type:=xlValue
    .DataLabel.Font.ColorIndex = 5
End With
```



# DataLabelRange Property

Returns a [Range](#) object that represents the range that contains the labels for the data fields in the PivotTable report. Read-only.

## Example

This example selects the data field labels in the PivotTable report.

```
Worksheets("Sheet1").Activate  
Range("A3").Select  
ActiveCell.PivotTable.DataLabelRange.Select
```



# DataPivotField Property

Returns a [PivotField](#) object that represents all the data fields in a PivotTable.  
Read-only.

*expression*.**DataPivotField**

*expression* Required. An expression that returns a [PivotTable](#) object.

## Example

This example moves the second **PivotItem** object to the first position. It assumes a PivotTable exists on the active worksheet and that the PivotTable contains data fields.

```
Sub UseDataPivotField()  
    Dim pvtTable As PivotTable  
    Set pvtTable = ActiveSheet.PivotTables(1)  
    ' Move second PivotItem to the first position in PivotTable.  
    pvtTable.DataPivotField.PivotItems(2).Position = 1  
End Sub
```



# DataRange Property

Returns a [Range](#) object as shown in the following table. Read-only.

<b>Object</b>	<b>Data range</b>
Data field	Data contained in the field
Row, column, or page field	Items in the field
Item	Data qualified by the item

## Example

This example selects the PivotTable items in the field named "REGION."

```
Set pvtTable = Worksheets("Sheet1").Range("A3").PivotTable  
Worksheets("Sheet1").Activate  
pvtTable.PivotFields("REGION").DataRange.Select
```



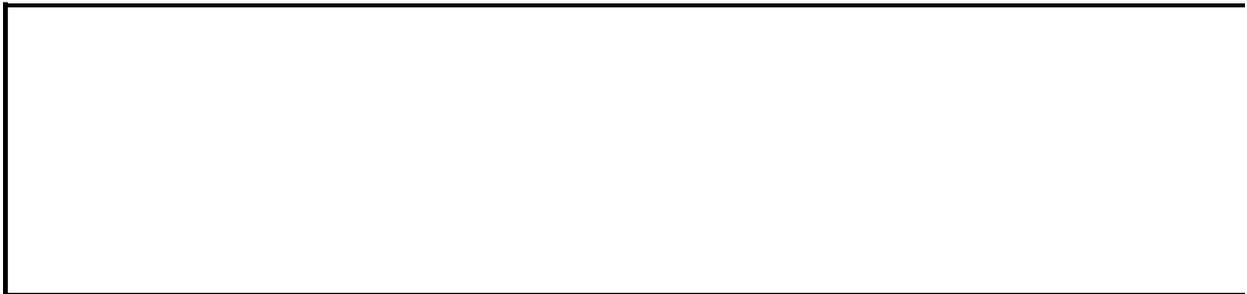
# DataTable Property

Returns a [DataTable](#) object that represents the chart data table. Read-only.

## Example

This example adds a data table with an outline border to the embedded chart.

```
With Worksheets(1).ChartObjects(1).Chart
    .HasDataTable = True
    .DataTable.HasBorderOutline = True
End With
```



[Show All](#)

# Data Type Property

 [Data Type property as it applies to the \*\*Parameter\*\* object.](#)

Returns or sets the data type of the specified query parameter. Read/write [xlParameterDataType](#).

xlParameterDataType can be one of these xlParameterDataType constants.

**xlParamTypeBinary**

**xlParamTypeChar**

**xlParamTypeDecimal**

**xlParamTypeFloat**

**xlParamTypeLongVarBinary**

**xlParamTypeNumeric**

**xlParamTypeSmallInt**

**xlParamTypeTimestamp**

**xlParamTypeUnknown**

**xlParamTypeVarChar**

**xlParamTypeBigInt**

**xlParamTypeBit**

**xlParamTypeDate**

**xlParamTypeDouble**

**xlParamTypeInteger**

**xlParamTypeLongVarChar**

**xlParamTypeReal**

**xlParamTypeTime**

**xlParamTypeTinyInt**

**xlParamTypeVarBinary**

**xlParamTypeWChar**

*expression*.**Data Type**

*expression* Required. An expression that returns one of the above objects.

 [DataType](#) property as it applies to the **PivotField** object.

Returns a constant describing the type of data in the PivotTable field. Read-only **[XlPivotFieldType](#)**.

XlPivotFieldType can be one of these XlPivotFieldType constants.

**xlDate**

**xlNumber**

**xlText**

*expression*.**DataType**

*expression* Required. An expression that returns one of the above objects.

## Example

This example displays the data type of the field named "ORDER\_DATE."

```
Set pvtTable = Worksheets("Sheet1").Range("A3").PivotTable
Select Case pvtTable.PivotFields("ORDER_DATE").DataType
    Case Is = xlText
        MsgBox "The field contains text data"
    Case Is = xlNumber
        MsgBox "The field contains numeric data"
    Case Is = xlDate
        MsgBox "The field contains date data"
End Select
```



# Date1904 Property

**True** if the workbook uses the 1904 date system. Read/write **Boolean**.

## Example

This example causes Microsoft Excel to use the 1904 date system for the active workbook.

```
ActiveWorkbook.Date1904 = True
```



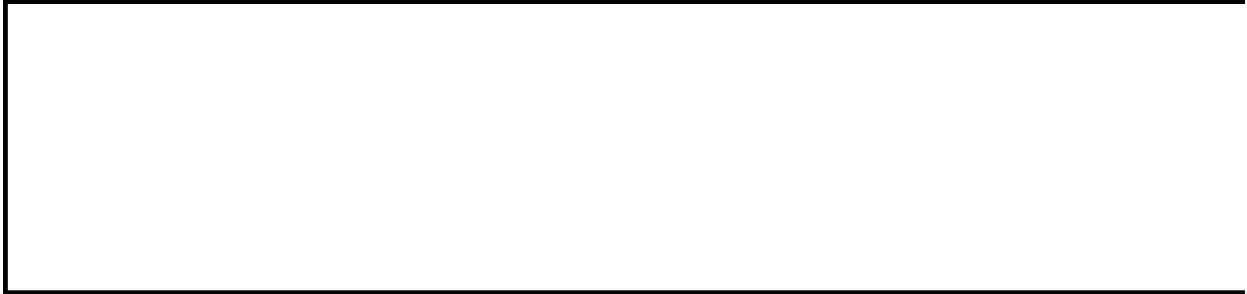
# **DDEAppReturnCode Property**

Returns the application-specific DDE return code that was contained in the last DDE acknowledge message received by Microsoft Excel. Read-only **Long**.

## Example

This example sets the variable `appErrorCode` to the DDE return code.

```
appErrorCode = Application.DDEAppReturnCode
```



[Show All](#)

# DecimalPlaces Property

Returns a **Long** value that represents the number of decimal places to show for the numbers in the **ListColumn** object. Returns 0 if the **ListDataFormat.Type** setting is not appropriate for decimal places. Returns **xlAutomatic** (-4105 decimal) if the Microsoft Windows SharePoint Services site is automatically determining the number of decimal places to show in the SharePoint list. Read-only **Long**.

*expression*.**DecimalPlaces**

*expression* Required. An expression that returns a **ListDataFormat** object.

## Remarks

In Excel, you cannot set any of the properties associated with the **ListDataFormat** object. You can set these properties, however, by modifying the list on the SharePoint site.

## Example

The following example returns the setting of the **DecimalPlaces** property for the third column of a [list](#) in Sheet1 of the active workbook.

```
Function GetDecimalPlaces() As Long
    Dim wrksht As Worksheet
    Dim objListCol As ListColumn

    Set wrksht = ActiveWorkbook.Worksheets("Sheet1")
    Set objListCol = wrksht.ListObjects(1).ListColumns(3)

    GetDecimalPlaces = objListCol.ListDataFormat.DecimalPlaces
End Function
```



# DecimalSeparator Property

Sets or returns the character used for the decimal separator as a **String**.  
Read/write.

*expression*.**DecimalSeparator**

*expression* Required. An expression that returns an **Application** object.

## Example

This example places "1,234,567.89" in cell A1 then changes the system separators to dashes for the decimals and thousands separators.

```
Sub ChangeSystemSeparators()  
  
    Range("A1").Formula = "1,234,567.89"  
    MsgBox "The system separators will now change."  
  
    ' Define separators and apply.  
    Application.DecimalSeparator = "-"  
    Application.ThousandsSeparator = "-"  
    Application.UseSystemSeparators = False  
  
End Sub
```



# DefaultFilePath Property

Returns or sets the default path that Microsoft Excel uses when it opens files.  
Read/write **String**.

## Example

This example displays the current default file path.

```
MsgBox "The current default file path is " & _  
    Application.DefaultFilePath
```



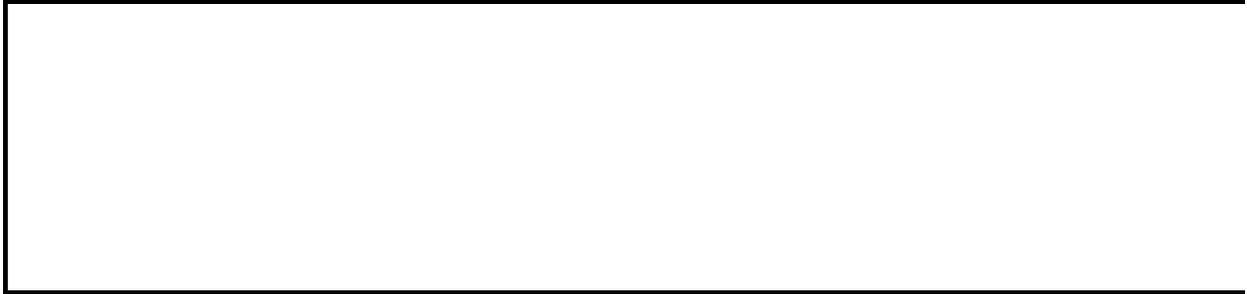
# DefaultSaveFormat Property

Returns or sets the default format for saving files. For a list of valid constants, see the [FileFormat](#) property. Read/write **Long**.

## Example

This example sets the default format for saving files.

```
Application.DefaultSaveFormat = xlExcel4Workbook
```



# DefaultSheetDirection Property

Returns or sets the default direction in which Microsoft Excel displays new windows and worksheets. Can be one of the following constants: **xIRTL** (right to left) or **xLTR** (left to right). Read/write **Long**.

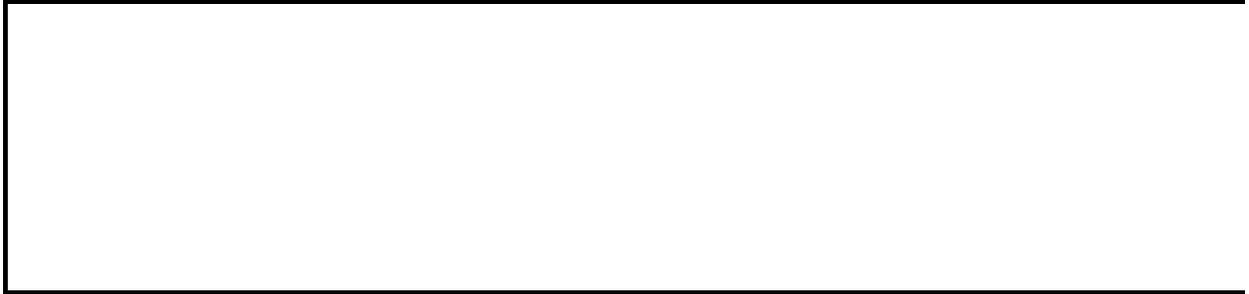
## Remarks

Some of these constants may not be available to you, depending on the language support (U.S. English, for example) that you've selected or installed.

## Example

This example sets right to left as the default direction.

```
Application.DefaultSheetDirection = xlRTL
```



[Show All](#)

# DefaultValue Property

Returns **Variant** representing the default data type value for a new row in a column. The **Nothing** object is returned when the schema does not specify a default value. Read-only **Variant**.

This property is used only for [lists](#) linked to a Microsoft Windows SharePoint Services site.

*expression*.**DefaultValue**

*expression* Required. An expression that returns a **ListDataFormat** object.

## Remarks

In Excel, you cannot set any of the properties associated with the **ListDataFormat** object. You can set these properties, however, by modifying the list on the SharePoint site.

## Example

The following example displays the setting of the **DefaultValue** property for the third column of the [list](#) in Sheet1 of the active workbook. This code requires a list linked to a SharePoint site.

```
Sub ShowDefaultSetting()  
    Dim wrksht As Worksheet  
    Dim objListCol As ListColumn  
  
    Set wrksht = ActiveWorkbook.Worksheets("Sheet1")  
    Set objListCol = wrksht.ListObjects(1).ListColumns(3)  
  
    If IsNull(objListCol.ListDataFormat.DefaultValue) Then  
        MsgBox "No default value specified."  
    Else  
        MsgBox objListCol.ListDataFormat.DefaultValue  
    End If  
End Sub
```



# DefaultWebOptions Property

Returns the [DefaultWebOptions](#) object that contains global application-level attributes used by Microsoft Excel whenever you save a document as a Web page or open a Web page. Read-only.

## Example

This example checks to see whether the default setting for document encoding is Western, and then it sets the string `strDocEncoding` accordingly.

```
If Application.DefaultWebOptions.Encoding = msoEncodingWestern Then
    strDocEncoding = "Western"
Else
    strDocEncoding = "Other"
End If
```



# Delivery Property

Returns or sets the routing delivery method. Can be one of the following **XlRoutingSlipDelivery** constants: **xlOneAfterAnother** or **xlAllAtOnce**.  
Read/write **Long**.

## Remarks

You cannot set this property if routing is in progress

## Example

This example sends Book1.xls to three recipients, one after another.

```
Workbooks("BOOK1.XLS").HasRoutingSlip = True
With Workbooks("BOOK1.XLS").RoutingSlip
    .Delivery = xlOneAfterAnother
    .Recipients = Array("Adam Bendel", _
        "Jean Selva", "Bernard Gabor")
    .Subject = "Here is BOOK1.XLS"
    .Message = "Here is the workbook. What do you think?"
End With
Workbooks("BOOK1.XLS").Route
```



# Dependents Property

Returns a [Range](#) object that represents the range containing all the dependents of a cell. This can be a multiple selection (a union of **Range** objects) if there's more than one dependent. Read-only **Range** object.

*expression*.**Dependents**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remark

**Note** The **Dependents** property only works on the active sheet and can not trace remote references.

## Example

This example selects the dependents of cell A1 on Sheet1.

```
Worksheets("Sheet1").Activate  
Range("A1").Dependents.Select
```



[Show All](#)

# Depth Property

 [Depth property as it applies to the \*\*ThreeDFormat\*\* object.](#)

For the **ThreeDFormat** object, returns or sets the depth of the shape's extrusion. Can be a value from – 600 through 9600 (positive values produce an extrusion whose front face is the original shape; negative values produce an extrusion whose back face is the original shape). Read/write **Single**.

*expression*.**Depth**

*expression* Required. An expression that returns a **ThreeDFormat** object.

 [Depth property as it applies to the \*\*TickLabels\*\* object.](#)

For the **TickLabels** object, returns the number of levels of category tick labels. Read-only **Long**.

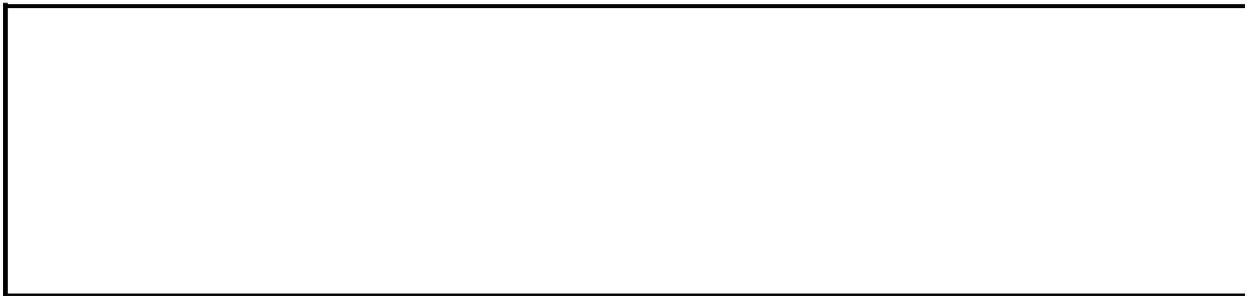
*expression*.**Depth**

*expression* Required. An expression that returns a **TickLabels** object.

## Example

This example adds an oval to myDocument and then specifies that the oval be extruded to a depth of 50 points and that the extrusion be purple.

```
Set myDocument = Worksheets(1)
Set myShape = myDocument.Shapes.AddShape(msoShapeOval, _
    90, 90, 90, 40)
With myShape.ThreeD
    .Visible = True
    .Depth = 50
    ' RGB value for purple
    .ExtrusionColor.RGB = RGB(255, 100, 255)
End With
```



# DepthPercent Property

Returns or sets the depth of a 3-D chart as a percentage of the chart width (between 20 and 2000 percent). Read/write **Long**.

## Example

This example sets the depth of Chart1 to be 50 percent of its width. The example should be run on a 3-D chart (the **DepthPercent** property fails on 2-D charts).

```
Charts("Chart1").DepthPercent = 50
```



# Destination Property

Returns the cell in the upper-left corner of the query table destination range (the range where the resulting query table will be placed). The destination range must be on the worksheet that contains the **QueryTable** object. Read-only **Range**.

## Example

This example scrolls through the active window until the upper-left corner of query table one is in the upper-left corner of the window.

```
Set d = Worksheets(1).QueryTables(1).Destination
With ActiveWindow
    .ScrollColumn = d.Column
    .ScrollRow = d.Row
End With
```



# Diagram Property

Returns a [Diagram](#) object representing a diagram.

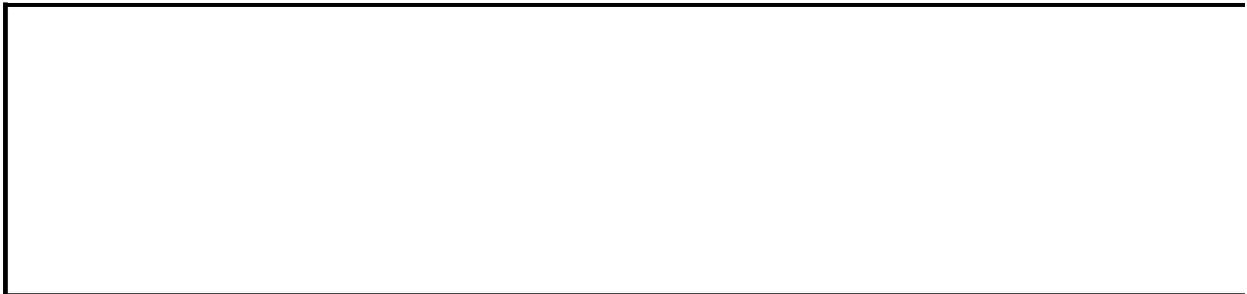
*expression*.**Diagram**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, an organization chart diagram is added to the active worksheet. Microsoft Excel then displays a message with the number of nodes added to the diagram.

```
Sub UseDiagram()  
  
    Dim wksOne As Worksheet  
    Dim shpOrgChart As Shape  
  
    Set wksOne = ActiveSheet  
  
    'Add organization chart diagram to current worksheet.  
    Set shpOrgChart = ActiveSheet.Shapes.AddDiagram _  
        (Type:=msoDiagramOrgChart, _  
         Top:=10, Left:=15, Width:=400, Height:=475)  
  
    'Add first node to organization chart.  
    shpOrgChart.DiagramNode.Children.AddNode  
  
    'Notify user of the number of nodes added to the diagram.  
    MsgBox shpOrgChart.Diagram.Nodes.Count  
  
End Sub
```



# DiagramNode Property

Returns a [DiagramNode](#) object that represents a node in a diagram.

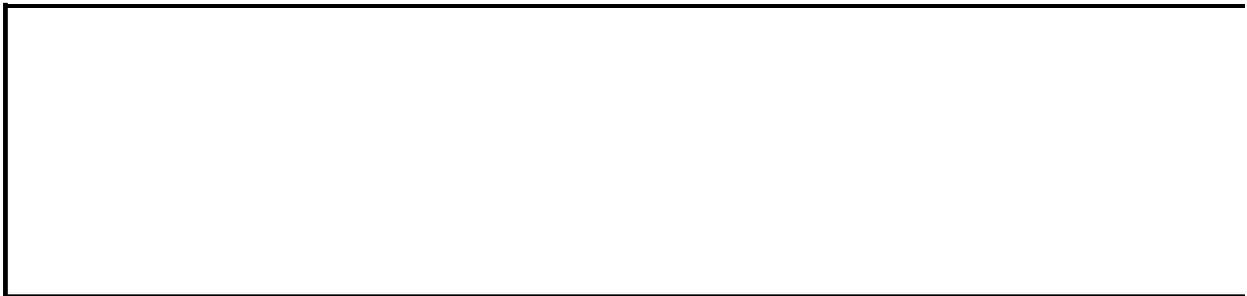
*expression*.**DiagramNode**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example adds a pyramid chart to the active worksheet and then adds four diagram nodes.

```
Sub CreatePyramidDiagram()  
  
    Dim dgnNode As DiagramNode  
    Dim shpDiagram As Shape  
    Dim intCount As Integer  
  
    'Add pyramid diagram to current document  
    Set shpDiagram = ActiveSheet.Shapes.AddDiagram( _  
        Type:=msoDiagramPyramid, Left:=10, _  
        Top:=15, Width:=400, Height:=475)  
  
    'Add first diagram node child  
    Set dgnNode = shpDiagram.DiagramNode.Children.AddNode  
  
    'Add three more diagram child nodes.  
    For intCount = 1 To 3  
        dgnNode.AddNode  
    Next intCount  
  
End Sub
```



# Dialogs Property

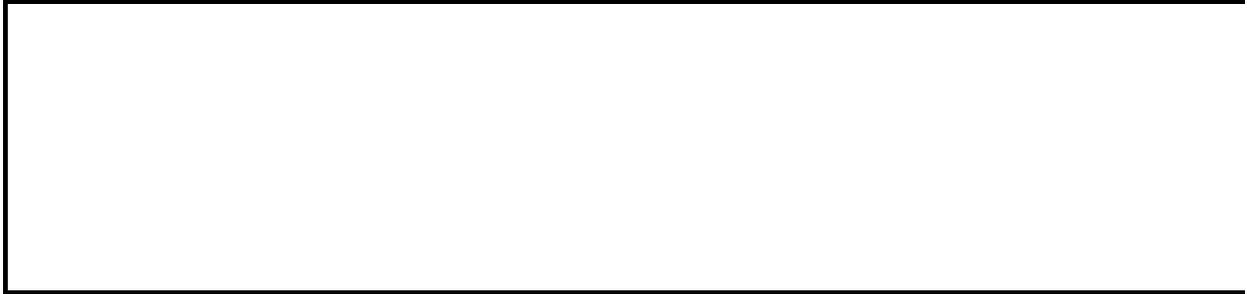
Returns a [Dialogs](#) collection that represents all built-in dialog boxes. Read-only.

For information about returning a single member of a collection, see [Returning an Object from a Collection](#).

## Example

This example displays the **Open** dialog box (**File** menu).

```
Application.Dialogs(xlDialogOpen).Show
```



# DictLang Property

Selects the dictionary language used when Microsoft Excel performs spelling checks. Read/write **Long**.

*expression*.**DictLang**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example sets the Excel dictionary to use the English (United States) language.

```
Sub LanguageSpellCheck()  
    With Application.SpellingOptions  
        .DictLang = 1033      ' United States English language number  
        .UserDict = "CUSTOM.DIC"  
    End With  
End Sub
```



# DirectDependents Property

Returns a [Range](#) object that represents the range containing all the direct dependents of a cell. This can be a multiple selection (a union of **Range** objects) if there's more than one dependent. Read-only **Range** object.

*expression*.**DirectDependents**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remark

**Note** The **Direct Dependents** property only works on the active sheet and can not trace remote references.

## Example

This example selects the direct dependents of cell A1 on Sheet1.

```
Worksheets("Sheet1").Activate  
Range("A1").DirectDependents.Select
```



[Show All](#)

# Direction Property

Returns or sets the order in which the cells will be spoken. The value of the **Direction** property is an [XISpeakDirection](#) constant. Read/write.

XISpeakDirection can be one of these XISpeakDirection constants.

**xlSpeakByColumns**

**xlSpeakByRows**

*expression*.**Direction**

*expression* Required. An expression that returns a **Speech** object.

## Example

In this example, Microsoft Excel determines the speech direction and notifies the user.

```
Sub CheckSpeechDirection()  
    ' Notify user of speech direction.  
    If Application.Speech.Direction = xlSpeakByColumns Then  
        MsgBox "The speech direction is set to speak by columns."  
    Else  
        MsgBox "The speech direction is set to speak by rows."  
    End If  
End Sub
```



# DirectPrecedents Property

Returns a [Range](#) object that represents the range containing all the direct precedents of a cell. This can be a multiple selection (a union of **Range** objects) if there's more than one precedent. Read-only **Range** object.

*expression*.**DirectPrecedents**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remark

**Note** The **DirectPrecedents** property only works on the active sheet and can not trace remote references.

## Example

This example selects the direct precedents of cell A1 on Sheet1.

```
Worksheets("Sheet1").Activate  
Range("A1").DirectPrecedents.Select
```



# DisplayAlerts Property

**True** if Microsoft Excel displays certain alerts and messages while a macro is running. Read/write **Boolean**.

## Remarks

The default value is **True**. Set this property to **False** if you don't want to be disturbed by prompts and alert messages while a macro is running; any time a message requires a response, Microsoft Excel chooses the default response.

If you set this property to **False**, Microsoft Excel sets this property to **True** when the code is finished, unless you are running cross process code.

When using the **SaveAs** method for workbooks to overwrite an existing file, the 'Overwrite' alert has a default of 'No', while the 'Yes' response is selected by Excel when the **DisplayAlerts** property is set equal to **True**.

## Example

This example closes the workbook Book1.xls and doesn't prompt the user to save changes. Any changes to Book1.xls aren't saved.

```
Application.DisplayAlerts = False  
Workbooks("BOOK1.XLS").Close  
Application.DisplayAlerts = True
```

This example suppresses the message that otherwise appears when you initiate a DDE channel to an application that's not running.

```
Application.DisplayAlerts = False  
channelNumber = Application.DDEInitiate( _  
    app:"WinWord", _  
    topic:"C:\WINWORD\FORMLETR.DOC")  
Application.DisplayAlerts = True  
Application.DDEExecute channelNumber, "[FILEPRINT]"  
Application.DDETerminate channelNumber  
Application.DisplayAlerts = True
```



# DisplayAutoCorrectOptions Property

Allows the user to display or hide the **AutoCorrect Options** button. The default value is **True**. Read/write **Boolean**.

*expression*.**DisplayAutoCorrectOptions**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

The **DisplayAutoCorrectOptions** property is a Microsoft Office-wide setting. Changing this property in Microsoft Excel will affect the other Office applications also.

In Excel the **AutoCorrect Options** button only appears when a hyperlink is automatically created.

## Example

This example determines if the **AutoCorrect Options** button can be displayed and notifies the user.

```
Sub CheckDisplaySetting()  
    'Determine setting and notify user.  
    If Application.AutoCorrect.DisplayAutoCorrectOptions = True Then  
        MsgBox "The AutoCorrect Options button can be displayed."  
    Else  
        MsgBox "The AutoCorrect Options button cannot be displayed."  
    End If  
End Sub
```



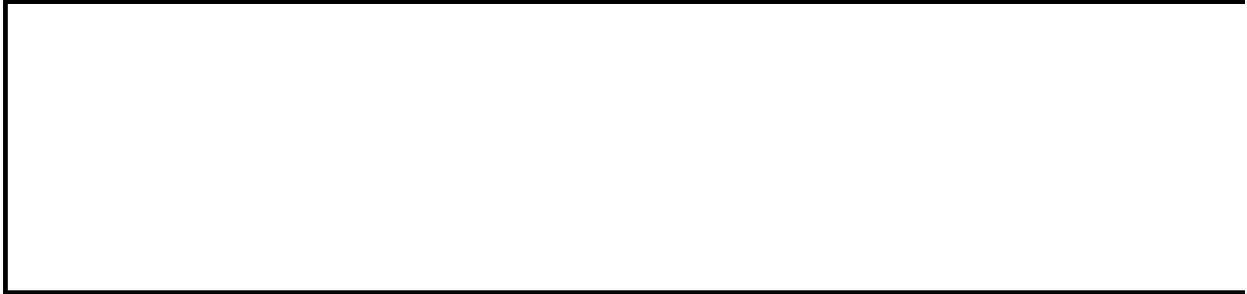
# DisplayBlanksAs Property

Returns or sets the way that blank cells are plotted on a chart. Can be one of the following **XlDisplayBlanksAs** constants: **xlNotPlotted**, **xlInterpolated**, or **xlZero**. Read/write **Long**.

## Example

This example sets Microsoft Excel to not plot blank cells in Chart1.

```
Charts("Chart1").DisplayBlanksAs = xlNotPlotted
```



# DisplayClipboardWindow Property

Returns **True** if the Microsoft Office Clipboard can be displayed. Read/write **Boolean**.

*expression*.**DisplayClipboardWindow**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel determines if the Office Clipboard can be displayed and notifies the user.

```
Sub SeeClipboard()  
    ' Determine if Office Clipboard can be displayed.  
    If Application.DisplayClipboardWindow = True Then  
        MsgBox "Office Clipboard can be displayed."  
    Else  
        MsgBox "Office Clipboard cannot be displayed."  
    End If  
End Sub
```



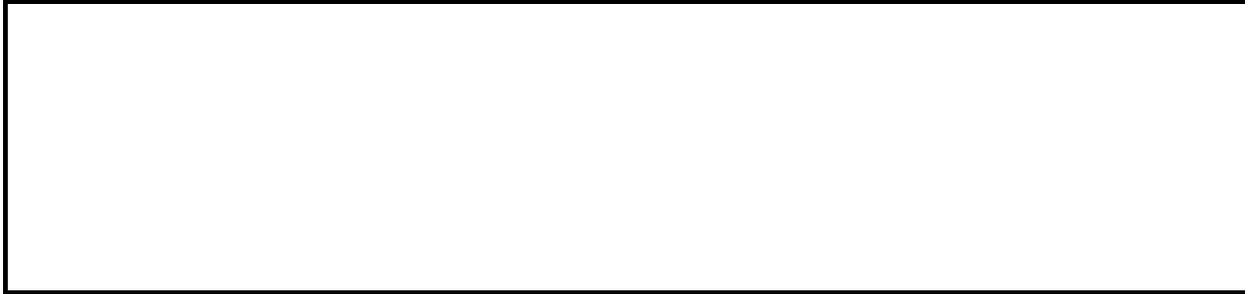
# DisplayCommentIndicator Property

Returns or sets the way cells display comments and indicators. Can be one of the following **XlCommentDisplayMode** constants: **xlNoIndicator**, **xlCommentIndicatorOnly**, or **xlCommentAndIndicator**. Read/write **Long**.

## Example

This example hides cell tips but retains comment indicators.

```
Application.DisplayCommentIndicator = xlCommentIndicatorOnly
```



# DisplayDocumentActionTaskPane Property

Set to **True** to display the **Document Actions** task pane; set to **False** to hide the **Document Actions** task pane. Read/write **Boolean**.

*expression*.**DisplayDocumentActionTaskPane**

*expression* Required. An expression that returns an [Application](#) object.

## Remarks

Setting this property to **True** will result in a run-time error if the active workbook is not a smart document.

--

# DisplayDrawingObjects Property

Returns or sets how shapes are displayed. Read/write **Long**.

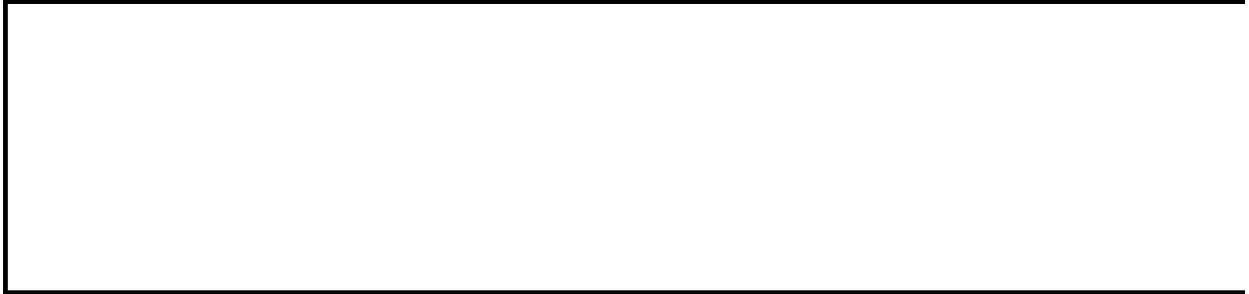
Can be one of the following **XIDisplayDrawingObjects** constants.

<b>Constant</b>	<b>Description</b>
<b>xIDisplayShapes</b>	Show all shapes.
<b>xIPlaceholders</b>	Show only placeholders.
<b>xIHide</b>	Hide all shapes.

## Example

This example hides all the shapes in the active workbook.

```
ActiveWorkbook.DisplayDrawingObjects = xlHide
```



[Show All](#)

# DisplayEmptyColumn Property

Returns **True** when the non-empty MDX keyword is included in the query to the OLAP provider for the value axis. The OLAP provider will not return empty columns in the result set. Returns **False** when the non-empty keyword is omitted. Read/write **Boolean**.

*expression*.**DisplayEmptyColumn**

*expression* Required. An expression that returns a **PivotTable** object.

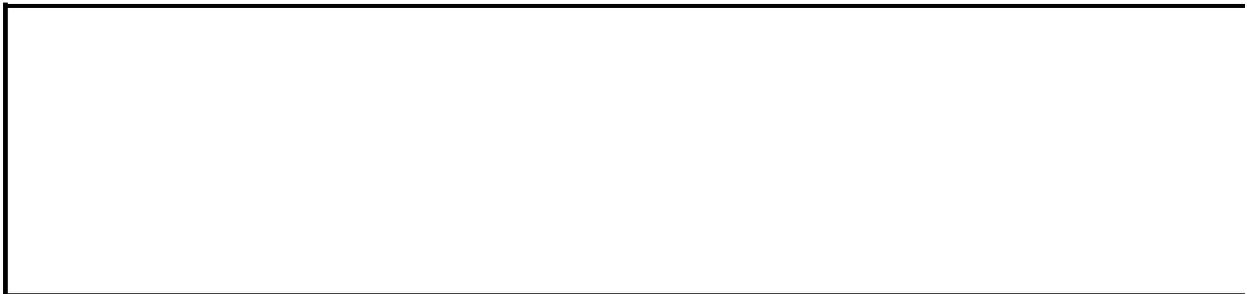
## Remarks

The PivotTable must be connected to an [Online Analytical Processing \(OLAP\)](#) data source to avoid a run-time error.

## Example

This example determines the display settings for empty columns in a PivotTable. It assumes a PivotTable connected to an OLAP data source exists on the active worksheet.

```
Sub CheckSetting()  
  
    Dim pvtTable As PivotTable  
  
    Set pvtTable = ActiveSheet.PivotTables(1)  
  
    ' Determine display setting for empty columns.  
    If pvtTable.DisplayEmptyColumn = False Then  
        MsgBox "Empty columns will not be displayed."  
    Else  
        MsgBox "Empty columns will be displayed."  
    End If  
  
End Sub
```



[Show All](#)

# DisplayEmptyRow Property

Returns **True** when the non-empty MDX keyword is included in the query to the OLAP provider for the category axis. The OLAP provider will not return empty rows in the result set. Returns **False** when the non-empty keyword is omitted.

Read/write **Boolean**.

*expression*.**DisplayEmptyRow**

*expression* Required. An expression that returns a **PivotTable** object.

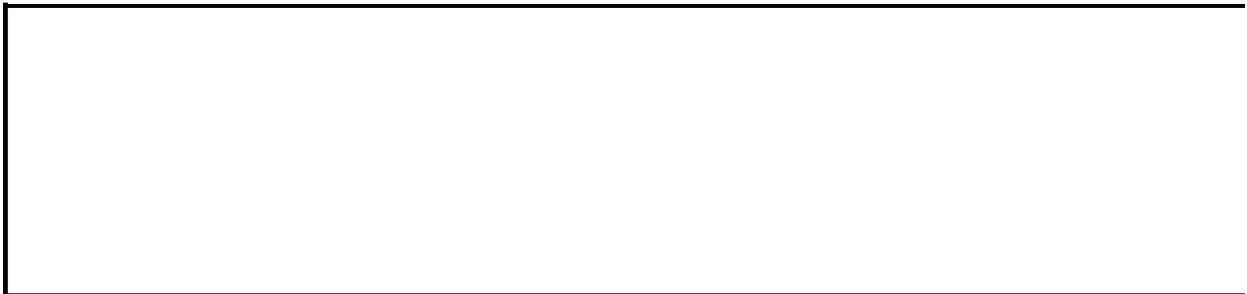
## Remarks

The PivotTable must be connected to an [Online Analytical Processing \(OLAP\)](#) data source to avoid a run-time error.

## Example

This example determines the display settings for empty rows in a PivotTable. It assumes a PivotTable connected to an OLAP data source exists on the active worksheet.

```
Sub CheckSetting()  
  
    Dim pvtTable As PivotTable  
  
    Set pvtTable = ActiveSheet.PivotTables(1)  
  
    ' Determine display setting for empty rows.  
    If pvtTable.DisplayEmptyRow = False Then  
        MsgBox "Empty rows will not be displayed."  
    Else  
        MsgBox "Empty rows will be displayed."  
    End If  
  
End Sub
```



# DisplayEquation Property

**True** if the equation for the trendline is displayed on the chart (in the same data label as the R-squared value). Setting this property to **True** automatically turns on data labels. Read/write **Boolean**.

## Example

This example displays the R-squared value and equation for trendline one in Chart1. The example should be run on a 2-D column chart that has a trendline for the first series.

```
With Charts("Chart1").SeriesCollection(1).Trendlines(1)  
    .DisplayRSquared = True  
    .DisplayEquation = True  
End With
```



# DisplayErrorString Property

**True** if the PivotTable report displays a custom error string in cells that contain errors. The default value is **False**. Read/write **Boolean**.

## Remarks

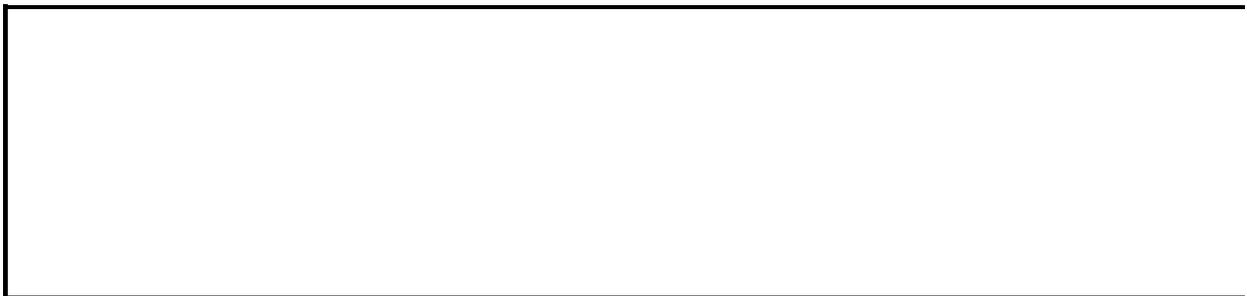
Use the **ErrorString** property to set the custom error string.

This property is particularly useful for suppressing divide-by-zero errors when calculated fields are pivoted.

## Example

This example causes the PivotTable report to display a hyphen in cells that contain errors.

```
With Worksheets(1).PivotTables("Pivot1")  
    .ErrorString = "-"  
    .DisplayErrorString = True  
End With
```



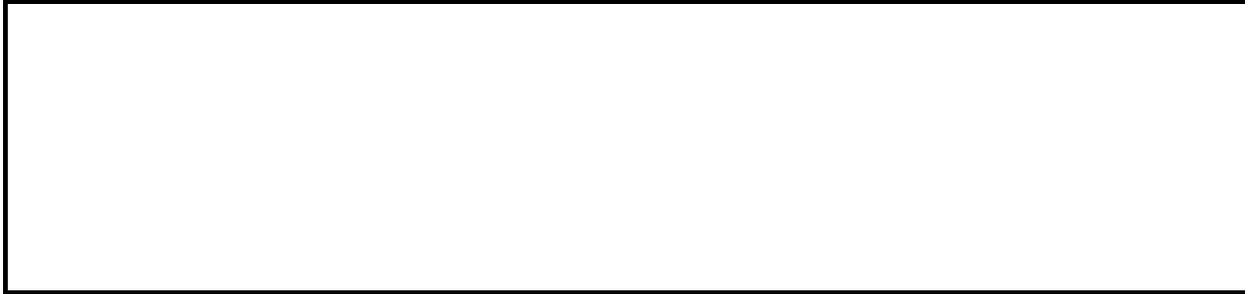
# DisplayExcel4Menus Property

**True** if Microsoft Excel displays version 4.0 menu bars. Read/write **Boolean**.

## Example

This example switches the display to Microsoft Excel version 4.0 menus.

```
Application.DisplayExcel4Menus = True
```



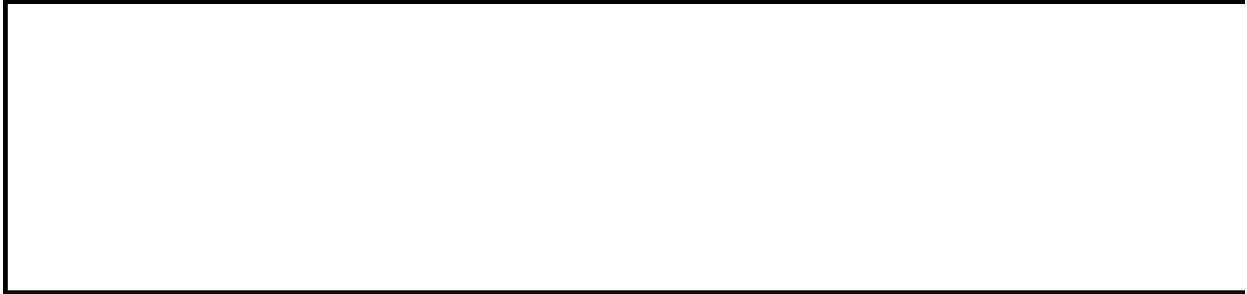
# DisplayFormulaBar Property

**True** if the formula bar is displayed. Read/write **Boolean**.

## Example

This example hides the formula bar.

```
Application.DisplayFormulaBar = False
```



# DisplayFormulas Property

**True** if the window is displaying formulas, **False** if the window is displaying values. Read/write **Boolean**.

## **Remarks**

This property applies only to worksheets and macro sheets.

## Example

This example changes the active window in Book1.xls to display formulas.

```
Workbooks("BOOK1.XLS").Worksheets("Sheet1").Activate  
ActiveWindow.DisplayFormulas = True
```



# DisplayFullScreen Property

**True** if Microsoft Excel is in full-screen mode. Read/write **Boolean**.

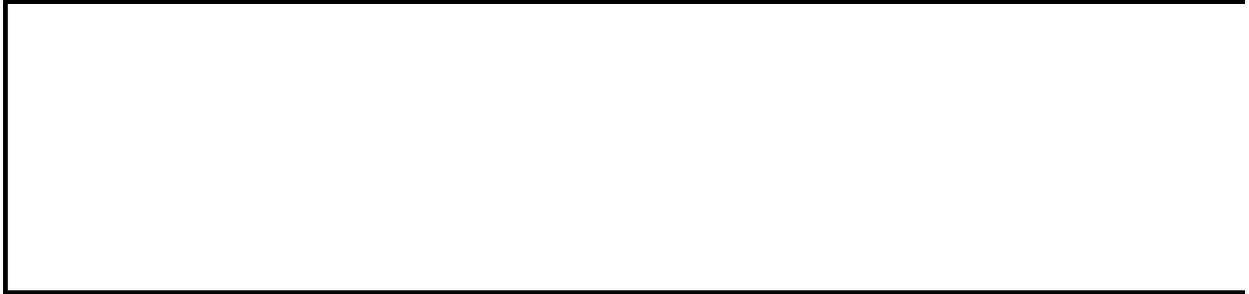
## **Remarks**

Full-screen mode maximizes the application window so that it fills the entire screen and hides the application title bar. Toolbars, the status bar, and the formula bar maintain separate display settings for full-screen mode and normal mode.

## Example

This example sets Microsoft Excel to be displayed in full-screen mode.

```
Application.DisplayFullScreen = True
```



# DisplayFunctionToolTips Property

**True** if function ToolTips can be displayed. Read/write **Boolean**.

*expression*.**DisplayFunctionToolTips**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel notifies the user the status of displaying function Tool Tips.

```
Sub CheckToolTip()  
    ' Notify the user of the ability to display function ToolTips.  
    If Application.DisplayFunctionToolTips = True Then  
        MsgBox "The ability to display function ToolTips is on."  
    Else  
        MsgBox "The ability to display function ToolTips is off."  
    End If  
End Sub
```



# DisplayGridlines Property

**True** if gridlines are displayed. Read/write **Boolean**.

## Remarks

This property applies only to worksheets and macro sheets.

This property affects only displayed gridlines. Use the [PrintGridlines](#) property to control the printing of gridlines.

## Example

This example toggles the display of gridlines in the active window in Book1.xls.

```
Workbooks("BOOK1.XLS").Worksheets("Sheet1").Activate  
ActiveWindow.DisplayGridlines = Not(ActiveWindow.DisplayGridlines)
```



# DisplayHeadings Property

**True** if both row and column headings are displayed, **False** if there are no headings displayed. Read/write **Boolean**.

## Remarks

This property applies only to worksheets and macro sheets.

This property affects only displayed headings. Use the [PrintHeadings](#) property to control the printing of headings.

## Example

This example turns off the display of row and column headings in the active window in Book1.xls.

```
Workbooks("BOOK1.XLS").Worksheets("Sheet1").Activate  
ActiveWindow.DisplayHeadings = False
```



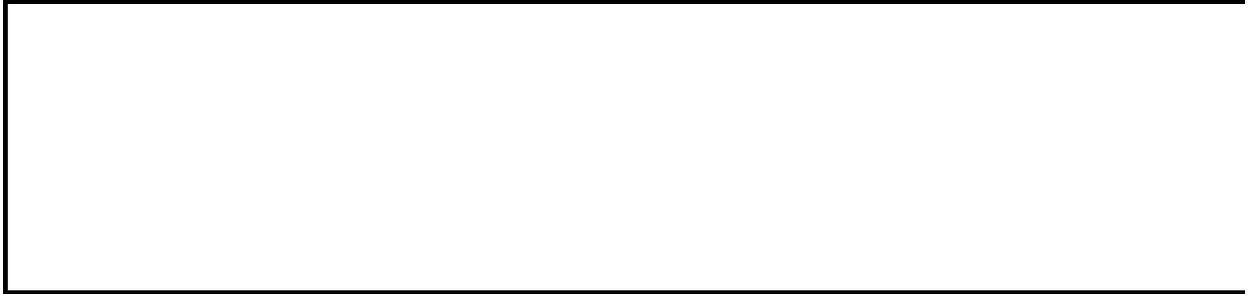
# DisplayHorizontalScrollBar Property

**True** if the horizontal scroll bar is displayed. Read/write **Boolean**.

## Example

This example turns on the horizontal scroll bar for the active window.

```
ActiveWindow.DisplayHorizontalScrollBar = True
```



# DisplayImmediateItems Property

Returns or sets a **Boolean** that indicates whether items in the row and column areas are visible when the data area of the PivotTable is empty. Set this property to **False** to hide the items in the row and column areas when the data area of the PivotTable is empty. The default value is **True**.

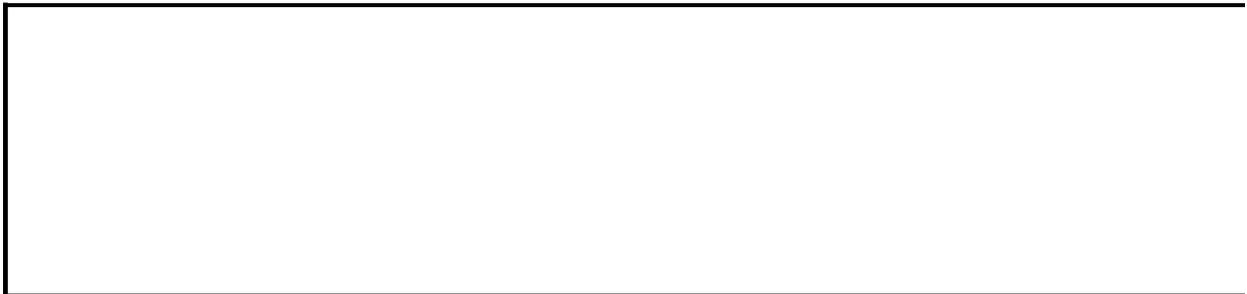
*expression*.**DisplayImmediateItems**

*expression* Required. An expression that returns a [PivotTable](#) object.

## Example

This example determines how the PivotTable was created and notifies the user. It assumes a PivotTable exists on the active worksheet.

```
Sub CheckItemsDisplayed()  
    Dim pvtTable As PivotTable  
  
    Set pvtTable = ActiveSheet.PivotTables(1)  
  
    ' Determine how the PivotTable was created.  
    If pvtTable.DisplayImmediateItems = True Then  
        MsgBox "Fields have been added to the row or column areas fo  
    Else  
        MsgBox "The PivotTable was created by using object-model cal  
    End If  
  
End Sub
```

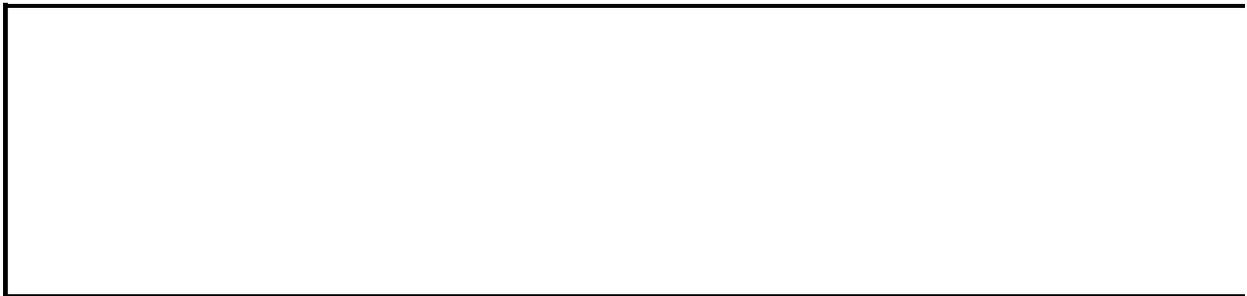


# DisplayInkComments Property

A **Boolean** value that determines whether ink comments are displayed in the workbook. Read/write **Boolean**.

*expression*.**DisplayInkComments**

*expression* Required. An expression that returns one of the objects in the Applies To list.



# DisplayInsertOptions Property

**True** if the **Insert Options** button should be displayed. Read/write **Boolean**.

*expression*.**DisplayInsertOptions**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel notifies the user the status of displaying the **Insert Options** button.

```
Sub SettingToolTip()  
    ' Notify the user of the ToolTip status.  
    If Application.DisplayInsertOptions = True Then  
        MsgBox "The ability to display the Insert Options button is  
    Else  
        MsgBox "The ability to display the Insert Options button is  
    End If  
End Sub
```



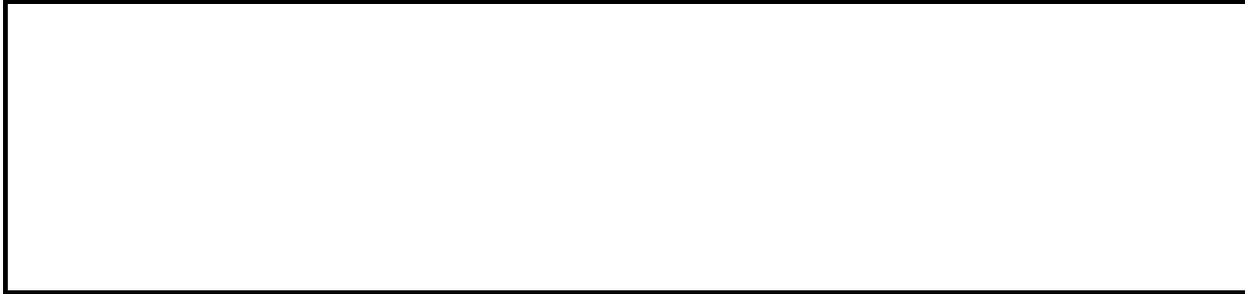
# DisplayNoteIndicator Property

**True** if cells containing notes display cell tips and contain note indicators (small dots in their upper-right corners). Read/write **Boolean**.

## Example

This example hides note indicators.

```
Application.DisplayNoteIndicator = False
```



# DisplayNullString Property

**True** if the PivotTable report displays a custom string in cells that contain null values. The default value is **True**. Read/write **Boolean**.

## Remarks

Use the **NullString** property to set the custom null string.

## Example

This example causes the PivotTable report to display "NA" in cells that contain null values.

```
With Worksheets(1).PivotTables("Pivot1")  
    .NullString = "NA"  
    .DisplayNullString = True  
End With
```

This example causes the PivotTable report to display 0 (zero) in cells that contain null values.

```
Worksheets(1).PivotTables("Pivot1").DisplayNullString = False
```



# DisplayOutline Property

**True** if outline symbols are displayed. Read/write **Boolean**.

## **Remarks**

This property applies only to worksheets and macro sheets.

## Example

This example displays outline symbols for the active window in Book1.xls.

```
Workbooks("BOOK1.XLS").Worksheets("Sheet1").Activate  
ActiveWindow.DisplayOutline = True
```



# DisplayPageBreaks Property

**True** if page breaks (both automatic and manual) on the specified worksheet are displayed. Read/write **Boolean**.

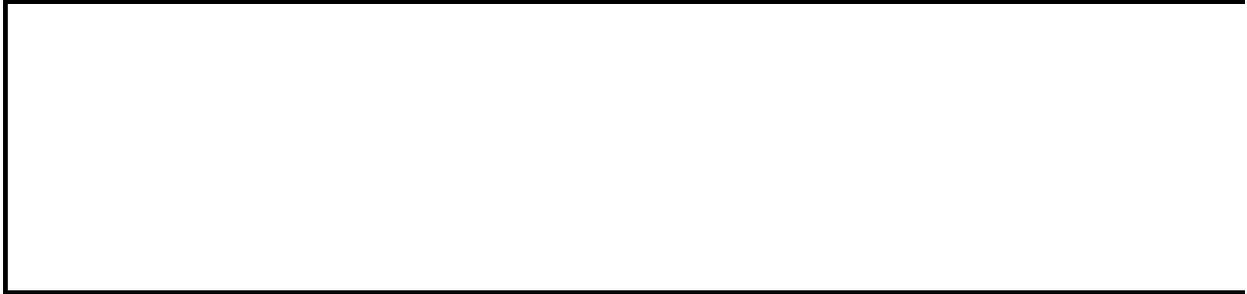
## Remarks

You can't set this property if you don't have a printer installed.

## Example

This example causes Sheet1 to display page breaks.

```
Worksheets("Sheet1").DisplayPageBreaks = True
```



# DisplayPasteOptions Property

**True** if the **Paste Options** button can be displayed. Read/write **Boolean**.

*expression*.**DisplayPasteOptions**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

This is a Microsoft Office-wide setting. This setting affects all other Microsoft Office applications. Setting the **DisplayPasteOptions** property to **True** turns off the **Auto Fill Options** button in Microsoft Excel. The **Auto Fill Options** button is only in Excel, but the **Paste Options** button is in all the other Microsoft Office applications.

## Example

In this example, Microsoft Excel notifies the user the status of displaying the **Paste Options** button.

```
Sub CheckDisplayFeature()  
    ' Check if the options button can be displayed.  
    If Application.DisplayPasteOptions = True Then  
        MsgBox "The ability to display the Paste Options button is o  
    Else  
        MsgBox "The ability to display the Paste Options button is o  
    End If  
End Sub
```



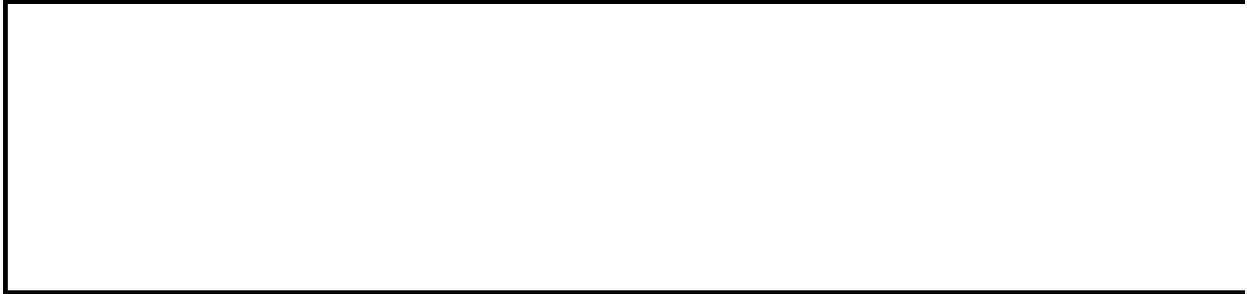
# DisplayRecentFiles Property

**True** if the list of recently used files is displayed on the **File** menu. Read/write **Boolean**.

## Example

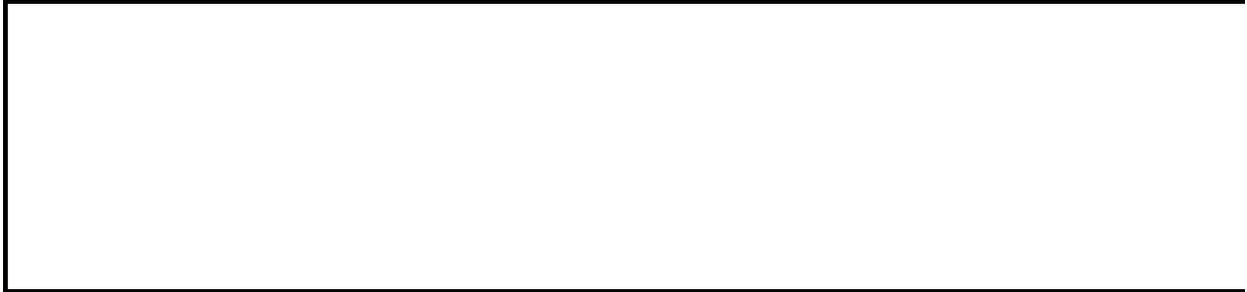
This example turns off the list of recently used files.

```
Application.DisplayRecentFiles = False
```



# DisplayRightToLeft Property

**True** if the specified window, worksheet, or **ListObject** is displayed from right to left instead of from left to right. **False** if the object is displayed from left to right. Read-only **Boolean**.



# DisplayRSquared Property

**True** if the R-squared value of the trendline is displayed on the chart (in the same data label as the equation). Setting this property to **True** automatically turns on data labels. Read/write **Boolean**.

## Example

This example displays the R-squared value and equation for trendline one in Chart1. The example should be run on a 2-D column chart that has a trendline for the first series.

```
With Charts("Chart1").SeriesCollection(1).Trendlines(1)  
    .DisplayRSquared = True  
    .DisplayEquation = True  
End With
```



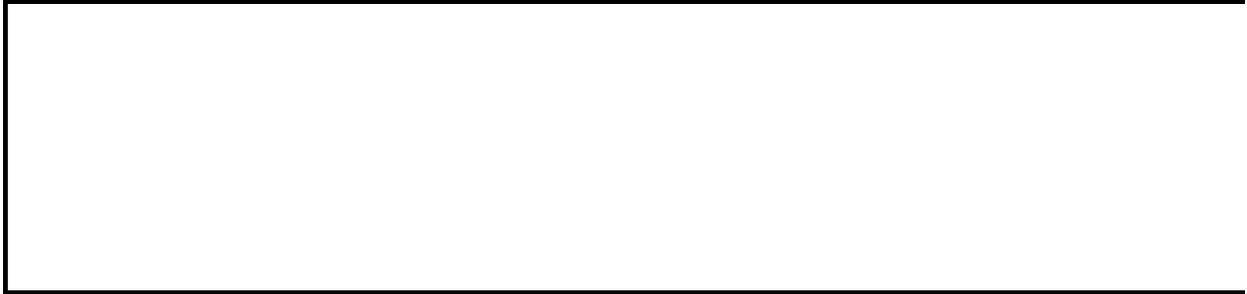
# DisplayScrollBars Property

**True** if scroll bars are visible for all workbooks. Read/write **Boolean**.

## Example

This example turns off scroll bars for all workbooks.

```
Application.DisplayScrollBars = False
```



[Show All](#)

# DisplaySmartTags Property

Returns or sets an [XLSmartTagDisplayMode](#) constant indicating the display features for smart tags. Read/write.

XLSmartTagDisplayMode can be one of these XLSmartTagDisplayMode constants.

**xlButtonOnly** Displays only the button for smart tags.

**xlDisplayNone** Nothing is displayed for smart tags.

**xlIndicatorAndButton** Display the indicator and button for smart tags.

*expression*.**DisplaySmartTags**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel determines the setting for displaying smart tags and notifies the user.

```
Sub CheckDisplayOptions()  
    ' Check the display options for smart tags.  
    Select Case ActiveWorkbook.SmartTagOptions.DisplaySmartTags  
        Case xlButtonOnly  
            MsgBox "The button for smart tags will only be displayed"  
        Case xlDisplayNone  
            MsgBox "Nothing will be displayed for smart tags."  
        Case xlIndicatorAndButton  
            MsgBox "The button and indicator will be displayed for s"  
    End Select  
End Sub
```



# DisplayStatusBar Property

**True** if the status bar is displayed. Read/write **Boolean**.

## Example

This example saves the current state of the **DisplayStatusBar** property and then sets the property to **True** so that the status bar is visible.

```
saveStatusBar = Application.DisplayStatusBar  
Application.DisplayStatusBar = True
```



[Show All](#)

# DisplayUnit Property

Returns or sets the unit label for the specified axis. Read/write [XIDisplayUnit](#).

XIDisplayUnit can be one of these XIDisplayUnit constants.

**xIHundredMillions**

**xIHundredThousands**

**xIMillions**

**xITenThousands**

**xIThousands**

**xIHundreds**

**xIMillionMillions**

**xITenMillions**

**xIThousandMillions**

*expression*.**DisplayUnit**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Using unit labels when charting large values makes your tick mark labels easier to read. For example, if you label your value axis in units of hundreds, thousands, or millions, you can use smaller numeric values at the tick marks on the axis.

## Example

This example sets the units displayed on the value axis in Chart1 to hundreds.

```
With Charts("Chart1").Axes(xlValue)  
    .DisplayUnit = xlHundreds  
    .HasTitle = True  
    .AxisTitle.Caption = "Rebate Amounts"  
End With
```



# DisplayUnitCustom Property

If the value of the [DisplayUnit](#) property is **xlCustom**, the **DisplayUnitCustom** property returns or sets the value of the displayed units. The value must be from 0 through 10E307. Read/write **Double**.

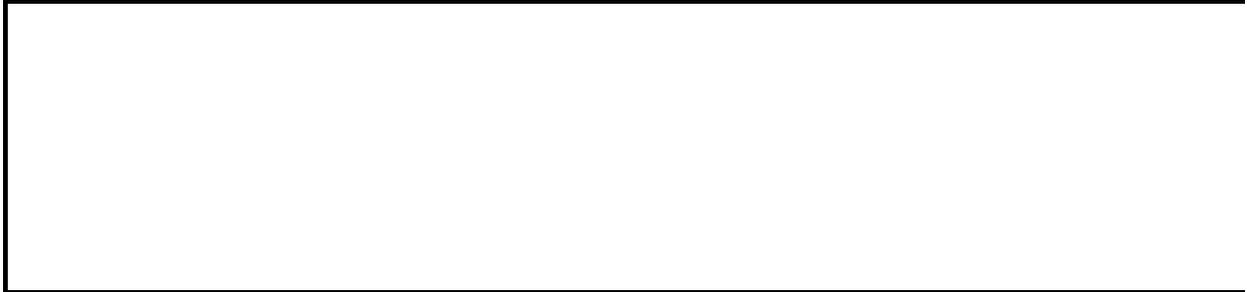
## Remarks

Using unit labels when charting large values makes your tick mark labels easier to read. For example, if you label your value axis in units of hundreds, thousands, or millions, you can use smaller numeric values at the tick marks on the axis.

## Example

This example sets the units displayed on the value axis in Chart1 to increments of 500.

```
With Charts("Chart1").Axes(xlValue)  
    .DisplayUnit = xlCustom  
    .DisplayUnitCustom = 500  
    .HasTitle = True  
    .AxisTitle.Caption = "Rebate Amounts"  
End With
```



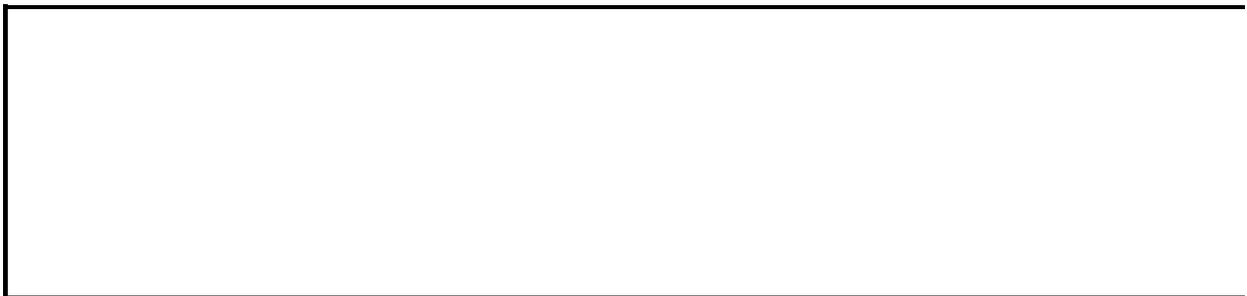
# DisplayUnitLabel Property

Returns the [DisplayUnitLabel](#) object for the specified axis. Returns **Null** if the [HasDisplayUnitLabel](#) property is set to **False**. Read-only.

## Example

This example sets the label caption to "Millions" for the value axis in Chart1, and then it turns off automatic font scaling.

```
With Charts("Chart1").Axes(xlValue).DisplayUnitLabel  
    .Caption = "Millions"  
    .AutoScaleFont = False  
End With
```



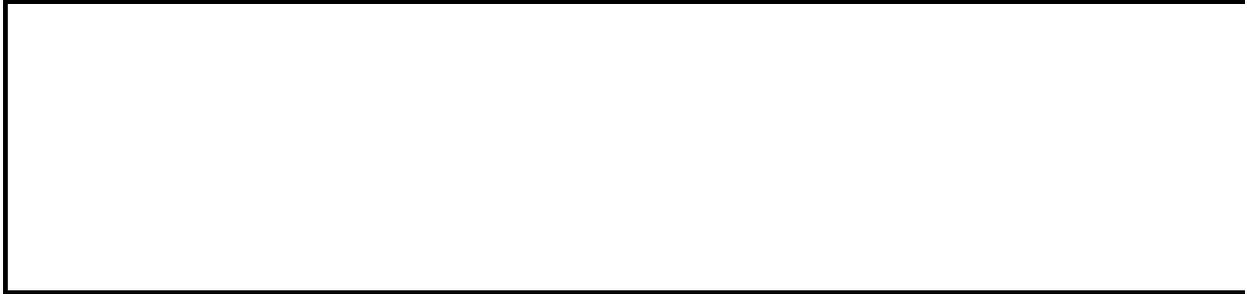
# DisplayVerticalScrollBar Property

**True** if the vertical scroll bar is displayed. Read/write **Boolean**.

## Example

This example turns on the vertical scroll bar for the active window.

```
ActiveWindow.DisplayVerticalScrollBar = True
```



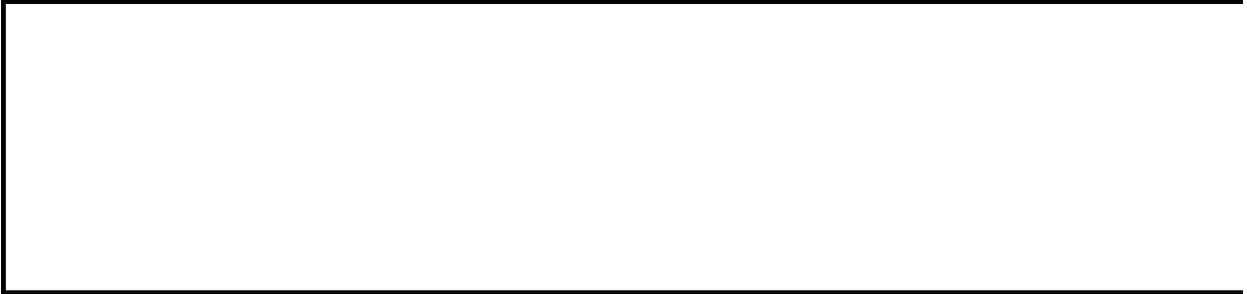
# DisplayWorkbookTabs Property

**True** if the workbook tabs are displayed. Read/write **Boolean**.

## Example

This example turns on the workbook tabs.

```
ActiveWindow.DisplayWorkbookTabs = True
```



# DisplayZeros Property

**True** if zero values are displayed. Read/write **Boolean**.

## **Remarks**

This property applies only to worksheets and macro sheets.

## Example

This example sets the active window in Book1.xls to display zero values.

```
Workbooks("BOOK1.XLS").Worksheets("Sheet1").Activate  
ActiveWindow.DisplayZeros = True
```



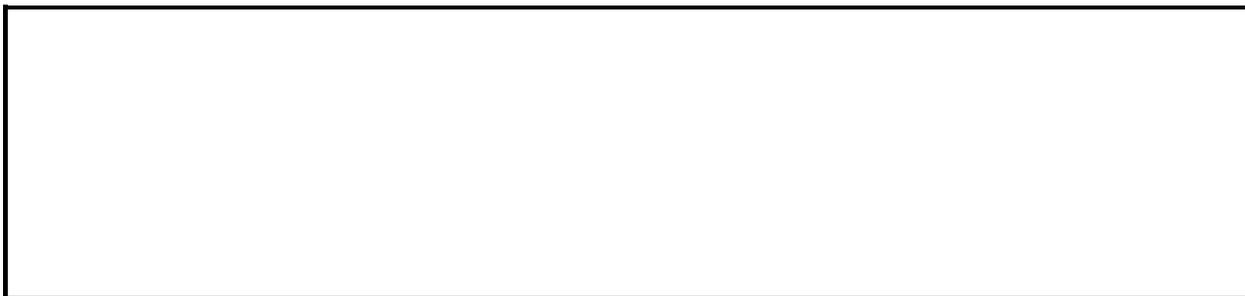
# DivID Property

Returns the unique identifier used for identifying an HTML <DIV> tag on a Web page. The tag is associated with an item in a document that you have saved to a Web page. An item can be an entire workbook, a worksheet, a selected print range, an AutoFilter range, a range of cells, a chart, a PivotTable report, or a query table. Read-only **String**.

## Example

This example saves a range of cells to a Web page, and then it obtains the identifier from the <DIV> tag of this item and finds the line on the saved Web page (q198.htm). The example also creates a copy of the Web page (newq1.htm) and inserts a comment line before the <DIV> tag in the copy of the file.

```
Set objP0 = ActiveWorkbook.PublishObjects.Add( _
    SourceType:=xlSourceRange, _
    Filename:="\\Server1\Reports\q198.htm", _
    Sheet:"Sheet1", _
    Source:="C2:D6", _
    HtmlType:=xlHtmlCalc)
objP0.Publish
strTargetDivID = objP0.DivID
Open "\\Server1\Reports\q198.htm" For Input As #1
Open "\\Server1\Reports\newq1.htm" For Output As #2
While Not EOF(1)
    Line Input #1, strFileLine
    If InStr(strFileLine, strTargetDivID) > 0 And _
        InStr(strFileLine, "<div") > 0 Then
        Print #2, "<!--Saved item-->"
    End If
    Print #2, strFileLine
Wend
Close #2
Close #1
```



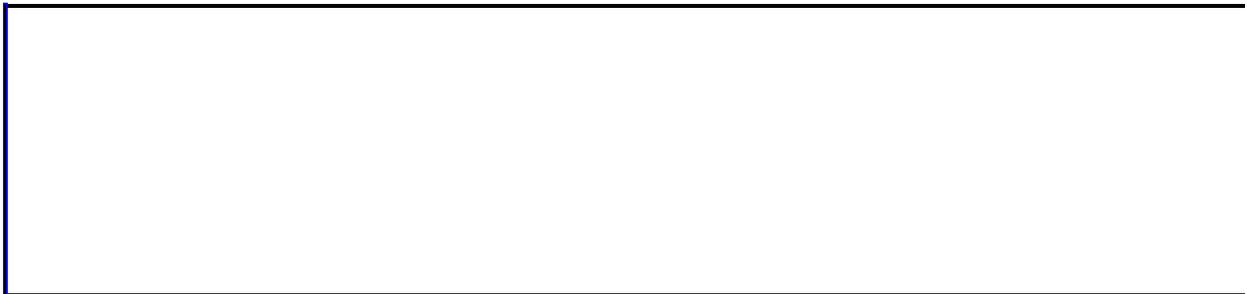
# DocumentLibraryVersions Property

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Returns a [DocumentLibraryVersions](#) collection that represents the collection of versions of a shared workbook that has versioning enabled and that is stored in a document library on a server.

*expression*.**DocumentLibraryVersions**

*expression* Required. An expression that returns a [Workbook](#) object.



# DoughnutHoleSize Property

Returns or sets the size of the hole in a doughnut chart group. The hole size is expressed as a percentage of the chart size, between 10 and 90 percent.

Read/write **Long**.

## Example

This example sets the hole size for doughnut group one in Chart1. The example should be run on a 2-D doughnut chart.

```
Charts("Chart1").DoughnutGroups(1).DoughnutHoleSize = 10
```



# DownBars Property

Returns a [DownBars](#) object that represents the down bars on a line chart.  
Applies only to line charts. Read-only.

## Example

This example turns on up bars and down bars for chart group one in Chart1 and then sets their colors. The example should be run on a 2-D line chart that has two series that cross each other at one or more data points.

```
With Charts("Chart1").ChartGroups(1)
    .HasUpDownBars = True
    .DownBars.Interior.ColorIndex = 3
    .UpBars.Interior.ColorIndex = 5
End With
```



# DownloadComponents Property

**True** if the necessary Microsoft Office Web components are downloaded when you view the saved document in a Web browser, but only if the components are not already installed. **False** if the components are not downloaded. The default value is **False**. Read/write **Boolean**.

## Remarks

You can set the [LocationOfComponents](#) property to a central URL (on the intranet or Web) or path (local or network) to a location from which authorized users can download components when viewing your saved document. The path must be valid and must point to a location that contains the necessary components, and the user must have a valid Microsoft Office 2000 license.

Office Web components add interactivity to documents that you save as Web pages. If you view a Web page in a browser on a computer that does not have the components installed, the interactive portions of the page will be static.

## Example

This example allows the Office Web components to be downloaded with the specified Web page, if they are not already installed.

```
Application.DefaultWebOptions.DownloadComponents = True  
Application.DefaultWebOptions.LocationOfComponents = _  
    Application.Path & Application.PathSeparator & "foo"
```



[Show All](#)

# DownloadURL Property

Returns a **String** representing the [Uniform Resource Locator \(URL\)](#) for a smart tag. Read-only.

*expression*.**DownloadURL**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

The URL address is specified in the related smart tag recognizer file. When a piece of text is recognized and marked, the URL becomes part of the information contained in the smart tag. The **DownloadURL** property is useful if a document is sent to someone who does not have the necessary recognizer and action files installed on their computer. The user can follow the URL to download the necessary smart tag files.

## Example

This example displays the URL for a smart tag.

```
Dim strURL String  
strURL = objSmartTag.DownloadURL
```



# Draft Property

**True** if the sheet will be printed without graphics. Read/write **Boolean**.

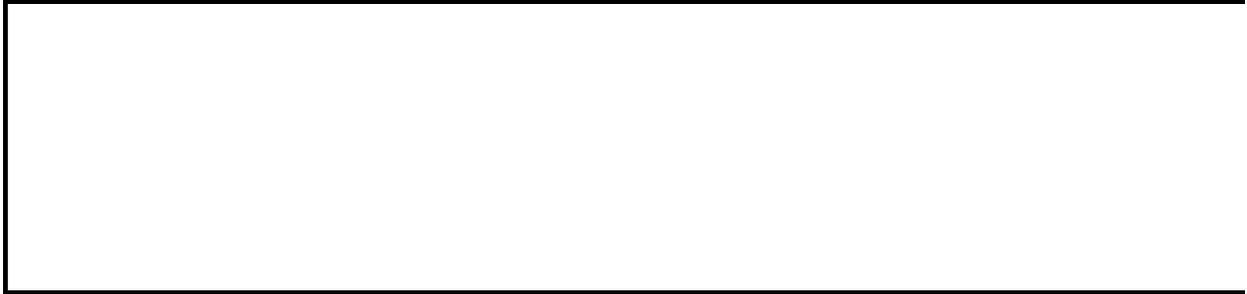
## Remarks

Setting this property to **True** makes printing faster (at the expense of not printing graphics).

## Example

This example turns off graphics printing for Sheet1.

```
Worksheets("Sheet1").PageSetup.Draft = True
```



[Show All](#)

# DragToColumn Property

**True** if the specified field can be dragged to the column position. The default value is **True**. Read/write **Boolean**.

## Remarks

For [OLAP](#) data sources, the value is **False** for measure fields.

## Example

This example prevents the Year field in the first PivotTable report on worksheet one from being dragged to the column position.

```
Worksheets(1).PivotTables("Pivot1") _  
    .PivotFields("Year").DragToColumn = False
```



[Show All](#)

# DragToData Property

**True** if the specified field can be dragged to the data position. The default value is **True**. Read/write **Boolean**.

## Remarks

For [OLAP](#) data sources, the value is **False** for measure fields.

## Example

This example prevents the Year field from being dragged to the data position in the first PivotTable report on the first worksheet.

```
Worksheets(1).PivotTables("Pivot1") _  
    .PivotFields("Year").DragToData = False
```



# DragToHide Property

**True** if the field can be hidden by being dragged off the PivotTable report. The default value is **True**. Read/write **Boolean**.

## Example

This example prevents the Year field in the first PivotTable report on worksheet one from being dragged off the report.

```
Worksheets(1).PivotTables("Pivot1") _  
    .PivotFields("Year").DragToHide = False
```



[Show All](#)

# DragToPage Property

**True** if the field can be dragged to the page position. The default value is **True**.  
Read/write **Boolean**.

## Remarks

For [OLAP](#) data sources, the value is **False** for measure fields.

## Example

This example prevents the Year field in the PivotTable report on worksheet one from being dragged to the page position.

```
Worksheets(1).PivotTables("Pivot1") _  
    .PivotFields("Year").DragToPage = False
```



[Show All](#)

# DragToRow Property

**True** if the field can be dragged to the row position. The default value is **True**.  
Read/write **Boolean**.

## Remarks

For [OLAP](#) data sources, the value is **False** for measure fields.

## Example

This example prevents the Year field in the first PivotTable report on worksheet one from being dragged to the row position.

```
Worksheets(1).PivotTables("Pivot1") _  
    .PivotFields("Year").DragToRow = False
```



[Show All](#)

# Drilled Property

Returns or sets the "drilled" (expanded, or visible) status of the [cube](#) field members in the hierarchical member-selection control of a cube field. This property is used primarily for macro recording and isn't intended for any other use. Read/write.

## Remarks

The **Drilled** property returns or sets an array. Each element of the array corresponds to a [level](#) of the cube field that has been expanded. The maximum number of elements is the number of levels in the cube field. Each element of the array is an array of type **String**, containing unique member names that are visible (expanded) at the corresponding level of the control. See the [TreeviewControl](#) object's [Hidden](#) property to determine when members are explicitly hidden in an expanded view.

## Example

This example expands the second-[level](#) members of the first [cube](#) field in the first PivotTable report on the active worksheet.

```
ActiveSheet.PivotTables("PivotTable1").CubeFields(1) _  
    .TreeviewControl.Drilled = _  
        Array(Array("", "", "", "", "", "", "", "", _  
            "", "", "", ""), _  
            Array("[state].[states].[AB]", _  
                "[state].[states].[CA]", _  
                "[state].[states].[IN]", _  
                "[state].[states].[KS]", _  
                "[state].[states].[KY]", _  
                "[state].[states].[MD]", _  
                "[state].[states].[MI]", _  
                "[state].[states].[OH]", _  
                "[state].[states].[OR]", _  
                "[state].[states].[TN]", _  
                "[state].[states].[UT]", _  
                "[state].[states].[WA]"))
```



[Show All](#)

# DrilledDown Property

**True** if the flag for the specified PivotTable field or PivotTable item is set to "drilled" (expanded, or visible). Read/write **Boolean**.

## Remarks

You can use this property only for [OLAP](#) data sources.

You cannot set this property if the field or item is hidden.

## Example

This example sets the flags to “not drilled” for all items in the state field in the third PivotTable report on the active worksheet.

```
ActiveSheet.PivotTables("PivotTable3") _  
    .PivotFields("state").DrilledDown = False
```



# Drop Property

For callouts with an explicitly set drop value, this property returns the vertical distance (in points) from the edge of the text bounding box to the place where the callout line attaches to the text box. This distance is measured from the top of the text box unless the **AutoAttach** property is set to **True** and the text box is to the left of the origin of the callout line (the place that the callout points to), in which case the drop distance is measured from the bottom of the text box. Read-only **Single**.

## Remarks

Use the [CustomDrop](#) method to set the value of this property.

The value of this property accurately reflects the position of the callout line attachment to the text box only if the callout has an explicitly set drop value — that is, if the value of the [DropType](#) property is **msoCalloutDropCustom**.

## Example

This example replaces the custom drop for shape one on myDocument with one of two preset drops, depending on whether the custom drop value is greater than or less than half the height of the callout text box. For the example to work, shape one must be a callout.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(1).Callout
    If .DropType = msoCalloutDropCustom Then
        If .Drop < .Parent.Height / 2 Then
            .PresetDrop msoCalloutDropTop
        Else
            .PresetDrop msoCalloutDropBottom
        End If
    End If
End With
```



# DropDownLines Property

Returns or sets the number of list lines displayed in the drop-down portion of a combo box. Read/write **Long**.

## Example

This example creates a combo box with 10 list lines.

```
With Worksheets(1).Shapes.AddFormControl(xlDropDown, _  
    Left:=10, Top:=10, Width:=100, Height:=10)  
    .ControlFormat.DropDownLines = 10  
End With
```



# DropLines Property

Returns a [DropLines](#) object that represents the drop lines for a series on a line chart or area chart. Applies only to line charts or area charts. Read-only.

## Example

This example turns on drop lines for chart group one in Chart1 and then sets their line style, weight, and color. The example should be run on a 2-D line chart that has one series.

```
With Charts("Chart1").ChartGroups(1)
    .HasDropLines = True
    With .DropLines.Border
        .LineStyle = xlThin
        .Weight = xlMedium
        .ColorIndex = 3
    End With
End With
```



[Show All](#)

# DropType Property

Returns a value that indicates where the callout line attaches to the callout text box. Read-only [MsoCalloutDropType](#).

MsoCalloutDropType can be one of these MsoCalloutDropType constants.

**msoCalloutDropCenter**

**msoCalloutDropMixed**

**msoCalloutDropBottom**

**msoCalloutDropCustom**

**msoCalloutDropTop**

*expression*.**DropType**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

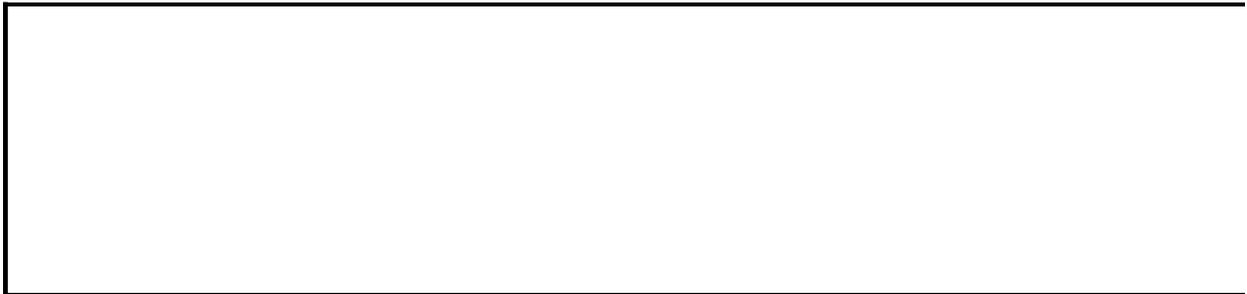
If the callout drop type is **msoCalloutDropCustom**, the values of the [Drop](#) and [AutoAttach](#) properties and the relative positions of the callout text box and callout line origin (the place that the callout points to) are used to determine where the callout line attaches to the text box.

This property is read-only. Use the [PresetDrop](#) method to set the value of this property.

## Example

This example replaces the custom drop for shape one on myDocument with one of two preset drops, depending on whether the custom drop value is greater than or less than half the height of the callout text box. For the example to work, shape one must be a callout.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(1).Callout
    If .DropType = msoCalloutDropCustom Then
        If .Drop < .Parent.Height / 2 Then
            .PresetDrop msoCalloutDropTop
        Else
            .PresetDrop msoCalloutDropBottom
        End If
    End If
End With
```



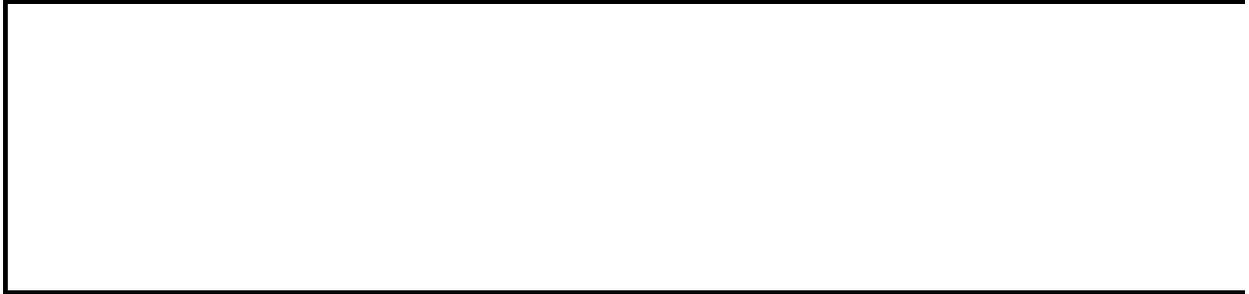
# EditDirectlyInCell Property

**True** if Microsoft Excel allows editing in cells. Read/write **Boolean**.

## Example

This example enables editing in cells.

```
Application.EditDirectlyInCell = True
```



[Show All](#)

# EditingType Property

If the specified node is a vertex, this property returns a value that indicates how changes made to the node affect the two segments connected to the node. Read-only [MsoEditingType](#).

MsoEditingType can be one of these MsoEditingType constants.

**msoEditingAuto**

**msoEditingCorner**

**msoEditingSmooth**

**msoEditingSymmetric**

*expression*.EditingType

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

This property is read-only. Use the [SetEditingType](#) method to set the value of this property.

## Example

This example changes all corner nodes to smooth nodes in shape three on myDocument. Shape three must be a freeform drawing.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(3).Nodes
    For n = 1 to .Count
        If .Item(n).EditingType = msoEditingCorner Then
            .SetEditingType n, msoEditingSmooth
        End If
    Next
End With
```



[Show All](#)

# EditWebPage Property

Returns or sets the web page [Uniform Resource Locator \(URL\)](#) for a web query.  
Read/write **Variant**.

*expression*.**EditWebPage**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

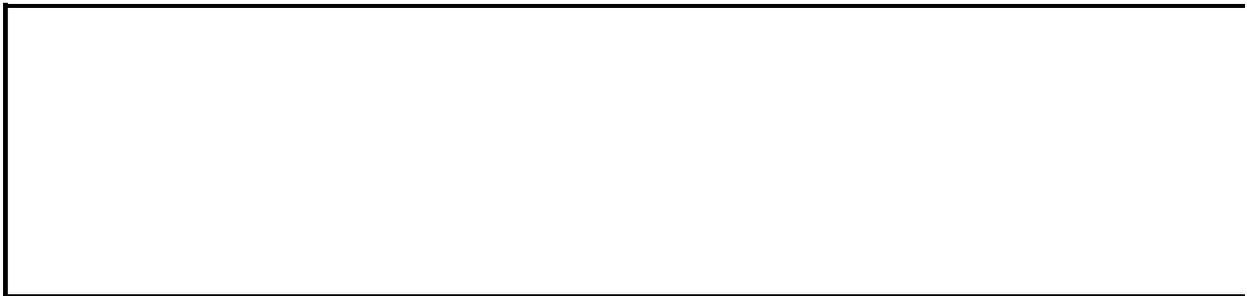
The **EditWebPage** property returns **Null** if not set. The **EditWebPage** property is only meaningful if the query type is Web or OLE.

If the **EditWebPage** is not null then ignore the **WebTables** property for refreshing. As a result an XML query and the **WebTable** property refers to the table in the original Web page and should only be used in the edit case to pre-populate the **Web Query** dialog box.

## Example

In this example, Microsoft Excel displays to the user a Web page URL. This example assumes a **QueryTable** object in cell A1 exists in the active worksheet and that a file called "MyHomepage.htm" exists on the C: drive.

```
Sub ReturnURL()  
  
    ' Set the EditWebPage property to a source.  
    Range("A1").QueryTable.EditWebPage = "C:\MyHomepage.htm"  
  
    ' Display the source to the user.  
    MsgBox Range("A1").QueryTable.EditWebPage  
  
End Sub
```



# Elevation Property

Returns or sets the elevation of the 3-D chart view, in degrees. Read/write **Long**.

## Remarks

The chart elevation is the height at which you view the chart, in degrees. The default is 15 for most chart types. The value of this property must be between -90 and 90, except for 3-D bar charts, where it must be between 0 and 44.

## Example

This example sets the chart elevation of Chart1 to 34 degrees. The example should be run on a 3-D chart (the **Elevation** property fails on 2-D charts).

```
Charts("Chart1").Elevation = 34
```



# EmailSubject Property

Returns or sets the text string of the specified hyperlink's e-mail subject line. The subject line is appended to the hyperlink's address. Read/write **String**.

## Remarks

This property is usually used with e-mail hyperlinks.

The value of this property takes precedence over any e-mail subject line you have specified by using the [Address](#) property of the same **Hyperlink** object.

## Example

This example sets the e-mail subject line for the first hyperlink in the first worksheet.

```
Worksheets(1).Hyperlinks(1).EmailSubject = "Quote Request"
```



# EmbedSmartTags Property

**True** to embed smart tags on the specified workbook. Read/write **Boolean**.

*expression*.**EmbedSmartTags**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel determines if smart tags are enabled for the active workbook and notifies the user.

```
Sub UseSmartTags()  
    ' Determine if smart tags are enabled for this workbook.  
    If ActiveWorkbook.SmartTagOptions.EmbedSmartTags = True Then  
        MsgBox "Smart tags can be embedded in this workbook."  
    Else  
        MsgBox "Smart tags can not be embedded in this workbook."  
    End If  
End Sub
```



# EmptyCellReferences Property

When set to **True** (default), Microsoft Excel identifies, with an **AutoCorrect Options** button, selected cells containing formulas that refer to empty cells. **False** disables empty cell reference checking. Read/write **Boolean**.

*expression*.**EmptyCellReferences**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In the following example, the **AutoCorrect Options** button appears for cell A1 which contains a formula that references empty cells.

```
Sub CheckEmptyCells()
```

```
    Application.ErrorCheckingOptions.EmptyCellReferences = True  
    Range("A1").Formula = "=A2+A3"
```

```
End Sub
```



# EnableAnimations Property

**True** if animated insertion and deletion is enabled. Read/write **Boolean**.

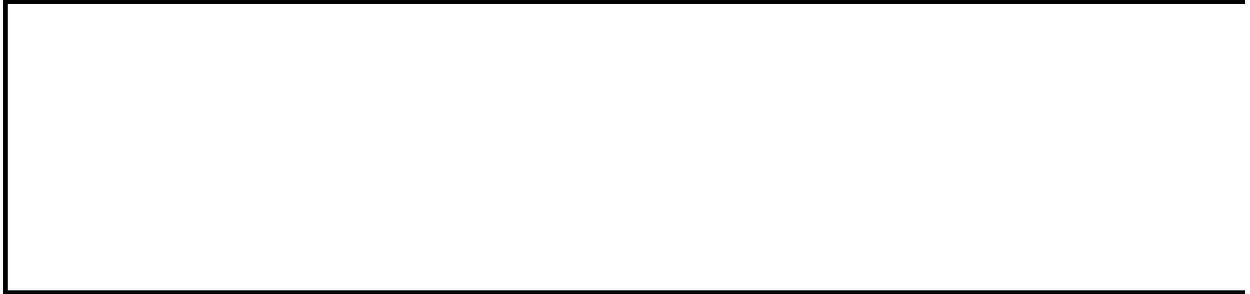
## **Remarks**

When animation is enabled, inserted worksheet rows and columns appear slowly, and deleted worksheet rows and columns disappear slowly.

## Example

This example turns off animated insertion and deletion.

```
Application.EnableAnimations = False
```



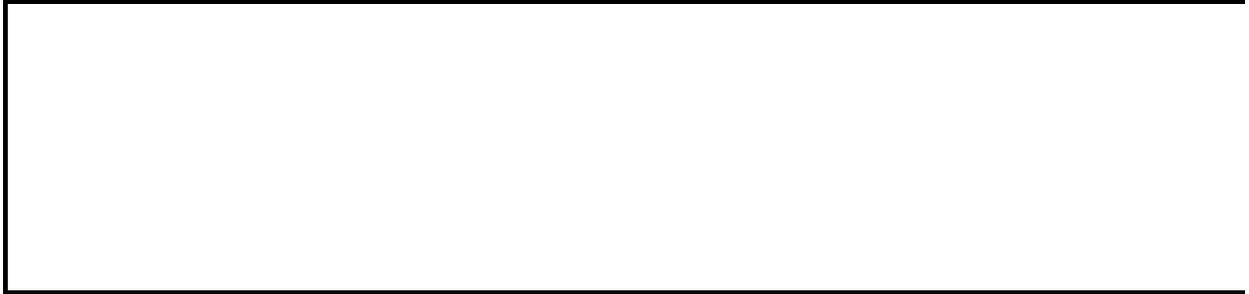
# EnableAutoComplete Property

**True** if the AutoComplete feature is enabled. Read/write **Boolean**.

## Example

This example enables the AutoComplete feature.

```
Application.EnableAutoComplete = True
```



# EnableAutoFilter Property

**True** if AutoFilter arrows are enabled when user-interface-only protection is turned on. Read/write **Boolean**.

## Remarks

This property applies to each worksheet and isn't saved with the worksheet or session.

## Example

This example enables the AutoFilter arrows on a protected worksheet.

```
ActiveSheet.EnableAutoFilter = True  
ActiveSheet.Protect contents:=True, userInterfaceOnly:=True
```



# EnableAutoRecover Property

Saves changed files, of all formats, on a timed interval. If Microsoft Excel fails, the system fails, or if the system is improperly shut down (not allowing Excel to save the changed files), the backed up files are opened and the user has an opportunity to save changes that otherwise would have been lost. When the user restarts Excel, a document recovery window opens, giving the user an option to recover the files they were working on. Setting this property to **True** (default) enables this feature. Read/write **Boolean**.

*expression*.**EnableAutoRecover**

*expression* Required. An expression that returns a [Workbook](#) object.

## Example

The following example checks the setting of the AutoRecover feature and if not enabled, Excel enables it and then notifies the user.

```
Sub UseAutoRecover()  
    ' Check to see if the feature is enabled, if not, enable it.  
    If ActiveWorkbook.EnableAutoRecover = False Then  
        ActiveWorkbook.EnableAutoRecover = True  
        MsgBox "The AutoRecover feature has been enabled."  
    Else  
        MsgBox "The AutoRecover feature is already enabled."  
    End If  
End Sub
```



# EnableCalculation Property

**True** if Microsoft Excel automatically recalculates the worksheet when necessary. **False** if Excel doesn't recalculate the sheet. Read/write **Boolean**.

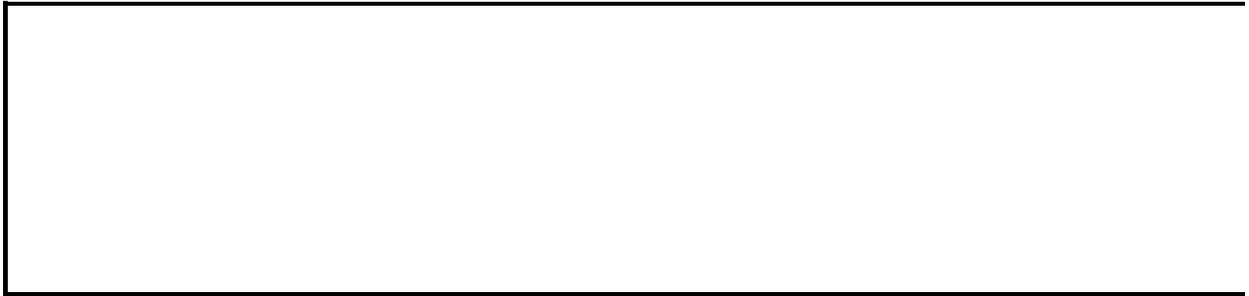
## Remarks

When the value of this property is **False**, you cannot request a recalculation. When you change the value from **False** to **True**, Excel recalculates the worksheet.

## Example

This example sets Microsoft Excel to not recalculate worksheet one automatically.

```
Worksheets(1).EnableCalculation = False
```



[Show All](#)

# EnableCancelKey Property

Controls how Microsoft Excel handles CTRL+BREAK (or ESC or COMMAND+PERIOD) user interruptions to the running procedure. Read/write [XlEnableCancelKey](#).

XlEnableCancelKey can be one of these XlEnableCancelKey constants.

**xlDisabled.** Cancel key trapping is completely disabled.

**xlErrorHandler.** The interrupt is sent to the running procedure as an error, trappable by an error handler set up with an **On Error GoTo** statement. The trappable error code is 18.

**xlInterrupt.** The current procedure is interrupted, and the user can debug or end the procedure.

*expression*.**EnableCancelKey**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Use this property very carefully. If you use **xlDisabled**, there's no way to interrupt a runaway loop or other non – self-terminating code. Likewise, if you use **xlErrorHandler** but your error handler always returns using the **Resume** statement, there's no way to stop runaway code.

The **EnableCancelKey** property is always reset to **xlInterrupt** whenever Microsoft Excel returns to the idle state and there's no code running. To trap or disable cancellation in your procedure, you must explicitly change the **EnableCancelKey** property every time the procedure is called.

## Example

This example shows how you can use the **EnableCancelKey** property to set up a custom cancellation handler.

```
On Error GoTo handleCancel
Application.EnableCancelKey = xlErrorHandler
MsgBox "This may take a long time: press ESC to cancel"
For x = 1 To 1000000      ' Do something 1,000,000 times (long!)
    ' do something here
Next x

handleCancel:
If Err = 18 Then
    MsgBox "You cancelled"
End If
```



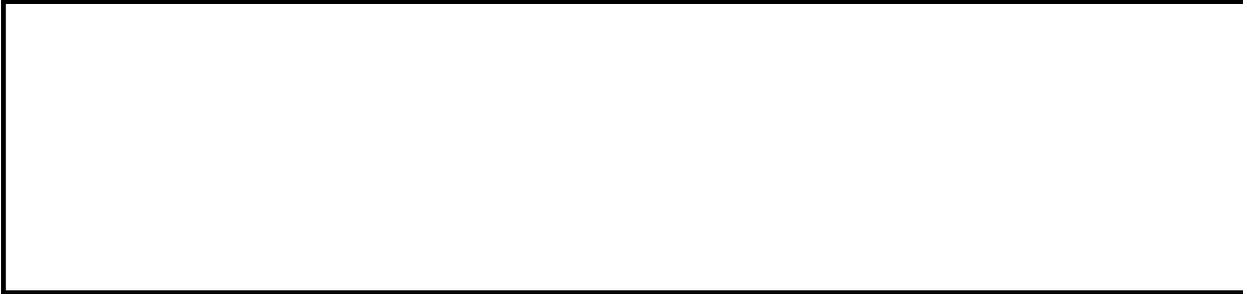
# Enabled Property

**True** if the object is enabled. Read/write **Boolean**.

## Example

This example disables embedded chart one on worksheet one.

```
Worksheets(1).ChartObjects(1).Enabled = False
```



# EnableDataValueEditing Property

**True** to disable the alert for when the user overwrites values in the data area of the PivotTable. **True** also allows the user to change data values that previously could not be changed. The default value is **False**. Read/write **Boolean**.

*expression.EnableDataValueEditing*

*expression* Required. An expression that returns a [PivotTable](#) object.

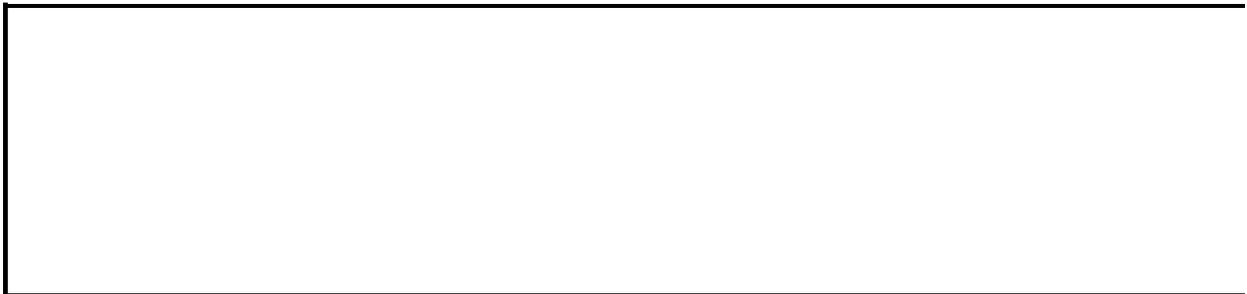
## **Remarks**

Any editing performed on data values is lost upon refresh.

## Example

This example determines the alert setting for overwriting values in the data area and notifies the user. The example assumes a PivotTable exists on the active worksheet.

```
Sub CheckAlertSetting()  
  
    Dim pvtTable As PivotTable  
  
    Set pvtTable = ActiveSheet.PivotTables(1)  
  
    ' Determine alert setting.  
    If pvtTable.EnableDataValueEditing = False Then  
        MsgBox "Alert is enabled."  
    Else  
        MsgBox "Alert is disabled."  
    End If  
  
End Sub
```



[Show All](#)

# EnableDrilldown Property

**True** if drilldown is enabled. The default value is **True**. Read/write **Boolean**.

## Remarks

Setting this property for a PivotTable report sets it for all fields in that report.

For [OLAP](#) data sources, the value is always **True**.

## Example

This example disables drilldown for all fields in the the first PivotTable report on worksheet one/.

```
Worksheets(1).PivotTables("Pivot1").EnableDrilldown = False
```



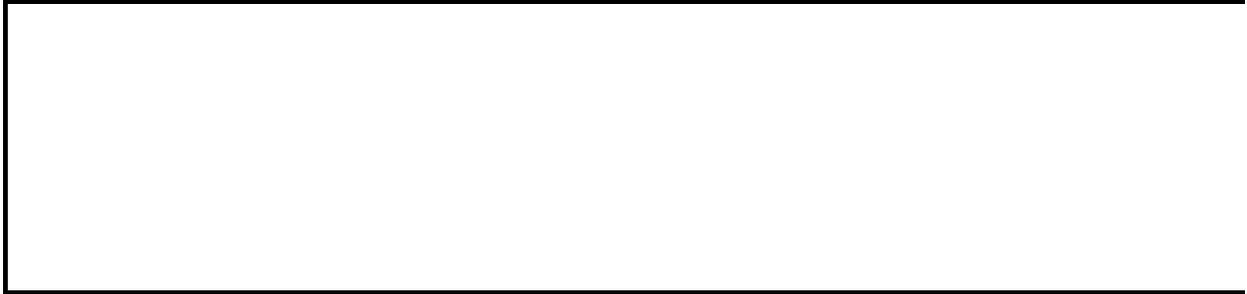
# EnableEditing Property

**True** if the user can edit the specified query table. **False** if the user can only refresh the query table. Read/write **Boolean**.

## Example

This example sets query table one so that the user cannot edit it.

```
worksheets(1).QueryTables(1).EnableEditing = False
```



# EnableEvents Property

**True** if events are enabled for the specified object. Read/write **Boolean**.

## Example

This example disables events before a file is saved so that the **BeforeSave** event doesn't occur.

```
Application.EnableEvents = False  
ActiveWorkbook.Save  
Application.EnableEvents = True
```



# EnableFieldDialog Property

**True** if the **PivotTable Field** dialog box is available when the user double-clicks the PivotTable field. The default value is **True**. Read/write **Boolean**.

## **Remarks**

Setting this property for a PivotTable report sets it for all fields in that report.

## Example

This example disables the **PivotTable Field** dialog box for the Year field.

```
Worksheets(1).PivotTables("Pivot1") _  
    .PivotFields("Year").EnableFieldDialog = False
```



# EnableFieldList Property

**False** to disable the ability to display the field list for the PivotTable. If the field list was already being displayed it disappears. The default value is **True**.  
Read/write **Boolean**.

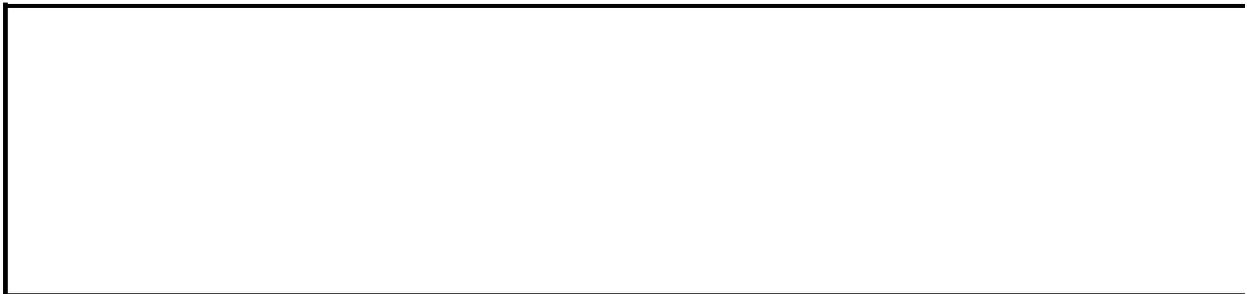
*expression*.**EnableFieldList**

*expression* Required. An expression that returns a [PivotTable](#) object.

## Example

This example determines the whether the field list for the PivotTable can be displayed or not and notifies the user. The example assumes that a PivotTable exists on the active worksheet.

```
Sub CheckFieldList()  
  
    Dim pvtTable As PivotTable  
  
    Set pvtTable = ActiveSheet.PivotTables(1)  
  
    ' Determine if field list can be displayed.  
    If pvtTable.EnableFieldList = True Then  
        MsgBox "The field list for the PivotTable can be displayed."  
    Else  
        MsgBox "The field list for the PivotTable cannot be displayed."  
    End If  
  
End Sub
```



# EnableItemSelection Property

When set to **False**, disables the ability to use the field dropdown in the user interface. The default value is **True**. Read/write **Boolean**.

*expression*.**EnableItemSelection**

*expression* Required. An expression that returns a [PivotField](#) object.

## **Remarks**

A run-time error will occur if the OLAP PivotTable field is not the highest level for the hierarchy.

## Example

This example determines the setting for selecting items using the field dropdown and enables the feature, if necessary. The example assumes a PivotTable exists on the active worksheet.

```
Sub UseEnableItemSelection()  
  
    Dim pvtTable As PivotTable  
    Dim pvtField As PivotField  
  
    Set pvtTable = ActiveSheet.PivotTables(1)  
    Set pvtField = pvtTable.RowFields(1)  
  
    ' Determine setting for property and enable if necessary.  
    If pvtField.EnableItemSelection = False Then  
        pvtField.EnableItemSelection = True  
        MsgBox "Item selection enabled for fields."  
    Else  
        MsgBox "Item selection is already enabled for fields."  
    End If  
  
End Sub
```



[Show All](#)

# EnableMultiplePageItems Property

**True** to allow multiple items in the page field area for OLAP PivotTables to be selected. The default value is **False**. Read/write **Boolean**.

*expression*.**EnableMultiplePageItems**

*expression* Required. An expression that returns a [CubeField](#) object.

## Remarks

This property only applies to [Online Analytical Processing \(OLAP\)](#) PivotTables. Querying or setting a non-OLAP PivotTable will result in a run-time error.

## Example

This example determines if multiple page items are enabled for the cube field and notifies the user. The example assumes that an OLAP PivotTable exists on the active worksheet.

```
Sub UseMultiplePageItems()  
  
    Dim pvtTable As PivotTable  
    Dim cbeField As CubeField  
  
    Set pvtTable = ActiveSheet.PivotTables(1)  
    Set cbeField = pvtTable.CubeFields("[Country]")  
  
    ' Determine setting for multiple page items.  
    If cbeField.EnableMultiplePageItems = False Then  
        MsgBox "Multiple page items cannot be selected."  
    Else  
        MsgBox "Multiple page items can be selected."  
    End If  
  
End Sub
```



# EnableOutlining Property

**True** if outlining symbols are enabled when user-interface-only protection is turned on. Read/write **Boolean**.

## Remarks

This property applies to each worksheet and isn't saved with the worksheet or session.

## Example

This example enables outlining symbols on a protected worksheet.

```
ActiveSheet.EnableOutlining = True  
ActiveSheet.Protect contents:=True, userInterfaceOnly:=True
```



# EnablePivotTable Property

**True** if PivotTable controls and actions are enabled when user-interface-only protection is turned on. Read/write **Boolean**.

## Remarks

This property applies to each worksheet and isn't saved with the worksheet or session.

There must be a sufficient number of unlocked cells below and to the right of the PivotTable report for Microsoft Excel to recalculate and display the PivotTable report.

## Example

This example enables PivotTable controls on a protected worksheet.

```
ActiveSheet.EnablePivotTable = True  
ActiveSheet.Protect contents:=True, userInterfaceOnly:=True
```



[Show All](#)

# EnableRefresh Property

**True** if the PivotTable cache or query table can be refreshed by the user. The default value is **True**. Read/write **Boolean**.

## Remarks

The **RefreshOnFileOpen** property is ignored if the **EnableRefresh** property is set to **False**.

For [OLAP](#) data sources, setting this property to **False** disables updates.

## Example

This example sets the first PivotTable report on worksheet one so that it cannot be refreshed.

```
Worksheets(1).PivotTables("Pivot1") _  
    .PivotCache.EnableRefresh = False
```



# EnableResize Property

**True** if the window can be resized. Read/write **Boolean**.

## Example

This example sets the active window so that it cannot be resized.

```
ActiveWindow.EnableResize = False
```



[Show All](#)

# EnableSelection Property

Returns or sets what can be selected on the sheet. Read/write [XlEnableSelection](#).

XlEnableSelection can be one of these XlEnableSelection constants.

**xlNoSelection**

**xlNoRestrictions**

**xlUnlockedCells**

*expression*.**EnableSelection**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

This property takes effect only when the worksheet is protected: **xlNoSelection** prevents any selection on the sheet, **xlUnlockedCells** allows only those cells whose **Locked** property is **False** to be selected, and **xlNoRestrictions** allows any cell to be selected.

## Example

This example sets worksheet one so that nothing on it can be selected.

```
With Worksheets(1)  
    .EnableSelection = xlNoSelection  
    .Protect Contents:=True, UserInterfaceOnly:=True  
End With
```



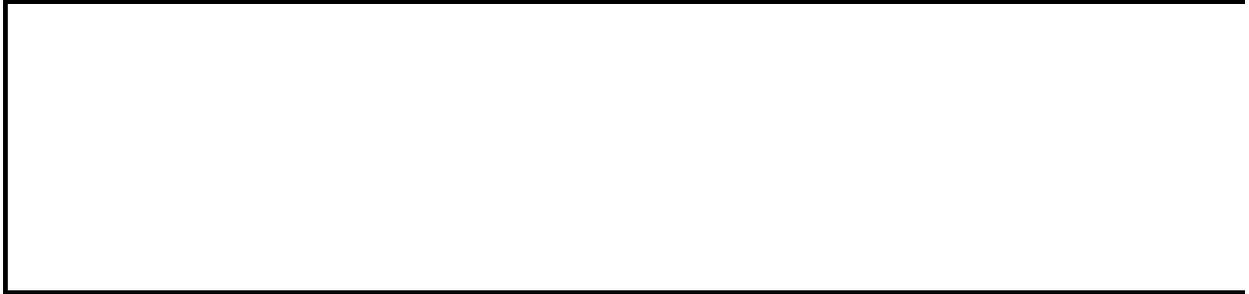
# EnableSound Property

**True** if sound is enabled for Microsoft Office. Read/write **Boolean**.

## Example

This example disables sound feedback.

```
Application.EnableSound = False
```



# EnableWizard Property

**True** if the PivotTable Wizard is available. The default value is **True**. Read/write **Boolean**.

## **Remarks**

When this property is set, the field wells aren't displayed on the worksheet.

## Example

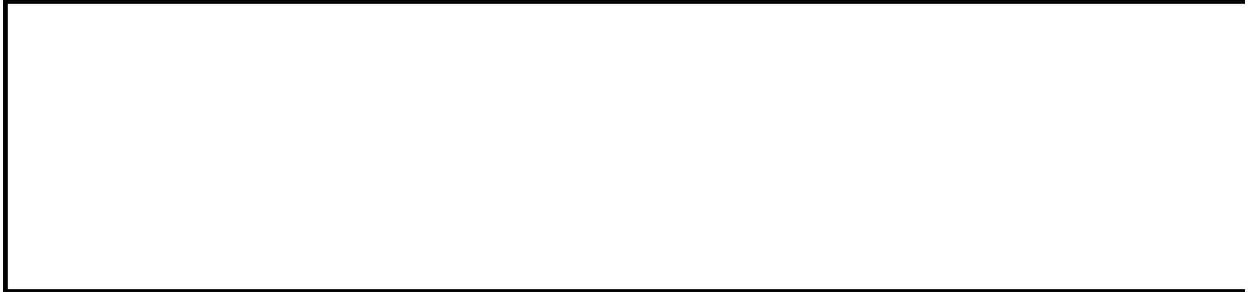
This example disables the PivotTable Wizard for the first PivotTable report on worksheet one.

```
Worksheets(1).PivotTables("Pivot1").EnableWizard = False
```



# Enclosures Property

You have requested Help for a Visual Basic keyword used only on the Macintosh. For information about this keyword, consult the language reference Help included with Microsoft Office Macintosh Edition.



[Show All](#)

# Encoding Property

Returns or sets the document encoding (code page or character set) to be used by the Web browser when you view the saved document. The default is the system code page. Read/write [MsoEncoding](#).

MsoEncoding can be one of these MsoEncoding constants.

**msoEncodingOEMMultilingualLatinI**

**msoEncodingOEMNordic**

**msoEncodingOEMTurkish**

**msoEncodingSimplifiedChineseAutoDetect**

**msoEncodingT61**

**msoEncodingTaiwanEten**

**msoEncodingTaiwanTCA**

**msoEncodingTaiwanWang**

**msoEncodingTraditionalChineseAutoDetect**

**msoEncodingTurkish**

**msoEncodingUnicodeLittleEndian**

**msoEncodingUTF7**

**msoEncodingVietnamese**

**msoEncodingEBCDICJapaneseKatakanaExtended**

**msoEncodingEBCDICJapaneseLatinExtendedAndJapanese**

**msoEncodingEBCDICKoreanExtendedAndKorean**

**msoEncodingEBCDICMultilingualROECELatin2**

**msoEncodingEBCDICSerbianBulgarian**

**msoEncodingEBCDICThai**

**msoEncodingEBCDICTurkishLatin5**

**msoEncodingEBCDICUSCanada**

**msoEncodingEBCDICUSCanadaAndTraditionalChinese**

**msoEncodingOEMModernGreek**

**msoEncodingOEMMultilingualLatinII**

**msoEncodingOEMPortuguese**

**msoEncodingOEMUnitedStates**  
**msoEncodingSimplifiedChineseGBK**  
**msoEncodingTaiwanCNS**  
**msoEncodingTaiwanIBM5550**  
**msoEncodingTaiwanTeleText**  
**msoEncodingThai**  
**msoEncodingTraditionalChineseBig5**  
**msoEncodingUnicodeBigEndian**  
**msoEncodingUSASCII**  
**msoEncodingUTF8**  
**msoEncodingWestern**  
**msoEncodingArabic**  
**msoEncodingArabicASMO**  
**msoEncodingArabicAutoDetect**  
**msoEncodingArabicTransparentASMO**  
**msoEncodingAutoDetect**  
**msoEncodingBaltic**  
**msoEncodingCentralEuropean**  
**msoEncodingCyrillic**  
**msoEncodingCyrillicAutoDetect**  
**msoEncodingEBCDICArabic**  
**msoEncodingEBCDICDenmarkNorway**  
**msoEncodingEBCDICFinlandSweden**  
**msoEncodingEBCDICFrance**  
**msoEncodingEBCDICGermany**  
**msoEncodingEBCDICGreek**  
**msoEncodingEBCDICGreekModern**  
**msoEncodingEBCDICHebrew**  
**msoEncodingEBCDICIcelandic**  
**msoEncodingEBCDICInternational**  
**msoEncodingEBCDICItaly**  
**msoEncodingEBCDICJapaneseKatakanaExtendedAndJapanese**  
**msoEncodingEBCDICKoreanExtended**

**msoEncodingEBCDICLatinAmericaSpain**  
**msoEncodingEBCDICRussian**  
**msoEncodingEBCDICSimplifiedChineseExtendedAndSimplifiedChinese**  
**msoEncodingEBCDICTurkish**  
**msoEncodingEBCDICUnitedKingdom**  
**msoEncodingEBCDICUSCanadaAndJapanese**  
**msoEncodingEUCChineseSimplifiedChinese**  
**msoEncodingEUCJapanese**  
**msoEncodingEUCKorean**  
**msoEncodingEUCTaiwaneseTraditionalChinese**  
**msoEncodingEuropa3**  
**msoEncodingExtAlphaLowercase**  
**msoEncodingGreek**  
**msoEncodingGreekAutoDetect**  
**msoEncodingHebrew**  
**msoEncodingHZGBSimplifiedChinese**  
**msoEncodingIA5German**  
**msoEncodingIA5IRV**  
**msoEncodingIA5Norwegian**  
**msoEncodingIA5Swedish**  
**msoEncodingISO2022CNSimplifiedChinese**  
**msoEncodingISO2022CNTraditionalChinese**  
**msoEncodingISO2022JPJISX02011989**  
**msoEncodingISO2022JPJISX02021984**  
**msoEncodingISO2022JPNoHalfwidthKatakana**  
**msoEncodingISO2022KR**  
**msoEncodingISO6937NonSpacingAccent**  
**msoEncodingISO885915Latin9**  
**msoEncodingISO88591Latin1**  
**msoEncodingISO88592CentralEurope**  
**msoEncodingISO88593Latin3**  
**msoEncodingISO88594Baltic**  
**msoEncodingISO88595Cyrillic**

**msoEncodingISO88596Arabic**  
**msoEncodingISO88597Greek**  
**msoEncodingISO88598Hebrew**  
**msoEncodingISO88599Turkish**  
**msoEncodingJapaneseAutoDetect**  
**msoEncodingJapaneseShiftJIS**  
**msoEncodingKOI8R**  
**msoEncodingKOI8U**  
**msoEncodingKorean**  
**msoEncodingKoreanAutoDetect**  
**msoEncodingKoreanJohab**  
**msoEncodingMacArabic**  
**msoEncodingMacCroatia**  
**msoEncodingMacCyrillic**  
**msoEncodingMacGreek1**  
**msoEncodingMacHebrew**  
**msoEncodingMacIcelandic**  
**msoEncodingMacJapanese**  
**msoEncodingMacKorean**  
**msoEncodingMacLatin2**  
**msoEncodingMacRoman**  
**msoEncodingMacRomania**  
**msoEncodingMacSimplifiedChineseGB2312**  
**msoEncodingMacTraditionalChineseBig5**  
**msoEncodingMacTurkish**  
**msoEncodingMacUkraine**  
**msoEncodingOEMArabic**  
**msoEncodingOEMBaltic**  
**msoEncodingOEMCanadianFrench**  
**msoEncodingOEMCyrillic**  
**msoEncodingOEMCyrillicII**  
**msoEncodingOEMGreek437G**  
**msoEncodingOEMHebrew**

## **msoEncodingOEMIcelandic**

### *expression*.**Encoding**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

You cannot use any of the constants that have the suffix **AutoDetect**. These constants are used by the [ReloadAs](#) method.

## Example

This example checks to see whether the default document encoding is Western, and then it sets the string `strDocEncoding` accordingly.

```
If Application.DefaultWebOptions.Encoding = msoEncodingWestern Then
    strDocEncoding = "Western"
Else
    strDocEncoding = "Other"
End If
```



[Show All](#)

# End Property

Returns a [Range](#) object that represents the cell at the end of the region that contains the source range. Equivalent to pressing END+UP ARROW, END+DOWN ARROW, END+LEFT ARROW, or END+RIGHT ARROW. Read-only **Range** object.

*expression*.**End**(*Direction*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*Direction* Required [XlDirection](#). The direction in which to move.

XlDirection can be one of these XlDirection constants.

**xlDown**

**xlToRight**

**xlToLeft**

**xlUp**

## Example

This example selects the cell at the top of column B in the region that contains cell B4.

```
Range("B4").End(xlUp).Select
```

This example selects the cell at the end of row 4 in the region that contains cell B4.

```
Range("B4").End(xlToRight).Select
```

This example extends the selection from cell B4 to the last cell in row four that contains data.

```
Worksheets("Sheet1").Activate  
Range("B4", Range("B4").End(xlToRight)).Select
```



[Show All](#)

# EndArrowheadLength Property

Returns or sets the length of the arrowhead at the end of the specified line.  
Read/write [MsoArrowheadLength](#).

MsoArrowheadLength can be one of these MsoArrowheadLength constants.

**msoArrowheadLengthMixed**

**msoArrowheadShort**

**msoArrowheadLengthMedium**

**msoArrowheadLong**

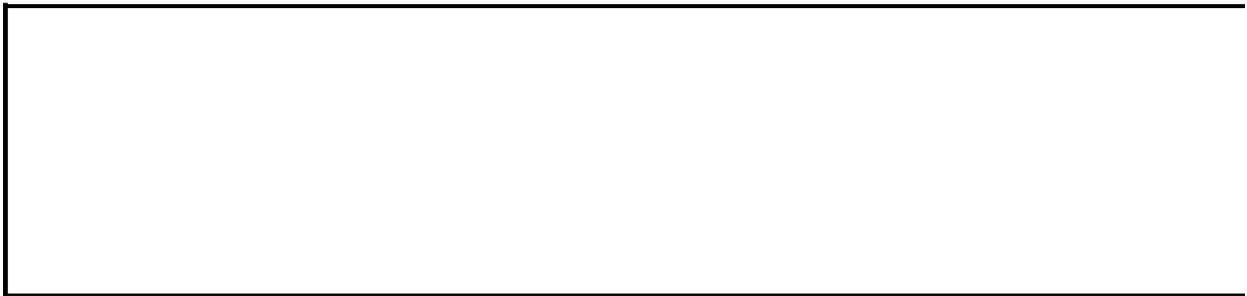
*expression*.**EndArrowheadLength**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example adds a line to myDocument. There's a short, narrow oval on the line's starting point and a long, wide triangle on its end point.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddLine(100, 100, 200, 300).Line
    .BeginArrowheadLength = msoArrowheadShort
    .BeginArrowheadStyle = msoArrowheadOval
    .BeginArrowheadWidth = msoArrowheadNarrow
    .EndArrowheadLength = msoArrowheadLong
    .EndArrowheadStyle = msoArrowheadTriangle
    .EndArrowheadWidth = msoArrowheadWide
End With
```



[Show All](#)

# EndArrowheadStyle Property

Returns or sets the style of the arrowhead at the end of the specified line.  
Read/write [MsoArrowheadStyle](#).

MsoArrowheadStyle can be one of these MsoArrowheadStyle constants.

**msoArrowheadNone**

**msoArrowheadOval**

**msoArrowheadStyleMixed**

**msoArrowheadDiamond**

**msoArrowheadOpen**

**msoArrowheadStealth**

**msoArrowheadTriangle**

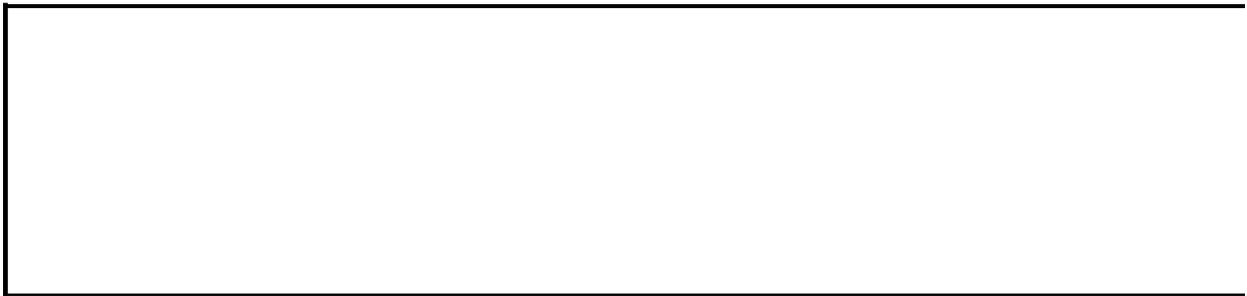
*expression*.**EndArrowheadStyle**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example adds a line to myDocument. There's a short, narrow oval on the line's starting point and a long, wide triangle on its end point.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddLine(100, 100, 200, 300).Line
    .BeginArrowheadLength = msoArrowheadShort
    .BeginArrowheadStyle = msoArrowheadOval
    .BeginArrowheadWidth = msoArrowheadNarrow
    .EndArrowheadLength = msoArrowheadLong
    .EndArrowheadStyle = msoArrowheadTriangle
    .EndArrowheadWidth = msoArrowheadWide
End With
```



[Show All](#)

# EndArrowheadWidth Property

Returns or sets the width of the arrowhead at the end of the specified line.  
Read/write [MsoArrowheadWidth](#).

MsoArrowheadWidth can be one of these MsoArrowheadWidth constants.

**msoArrowheadNarrow**

**msoArrowheadWidthMedium**

**msoArrowheadWide**

**msoArrowheadWidthMixed**

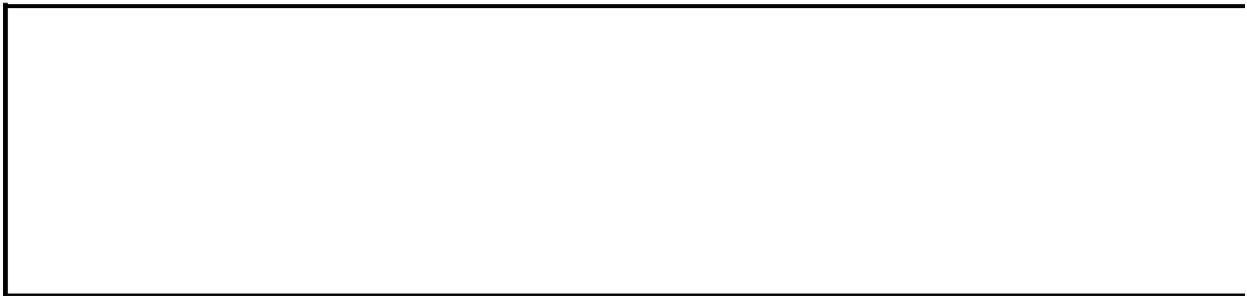
*expression*.**EndArrowheadWidth**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example adds a line to myDocument. There's a short, narrow oval on the line's starting point and a long, wide triangle on its end point.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddLine(100, 100, 200, 300).Line
    .BeginArrowheadLength = msoArrowheadShort
    .BeginArrowheadStyle = msoArrowheadOval
    .BeginArrowheadWidth = msoArrowheadNarrow
    .EndArrowheadLength = msoArrowheadLong
    .EndArrowheadStyle = msoArrowheadTriangle
    .EndArrowheadWidth = msoArrowheadWide
End With
```



[Show All](#)

# EndConnected Property

**msoTrue** if the end of the specified connector is connected to a shape. Read-only [MsoTriState](#).

MsoTriState can be one of these MsoTriState constants.

**msoCTrue** Does not apply to this property.

**msoFalse** The end of the specified connector is not connected to a shape.

**msoTriStateMixed** Does not apply to this property.

**msoTriStateToggle** Does not apply to this property.

**msoTrue** The end of the specified connector is connected to a shape.

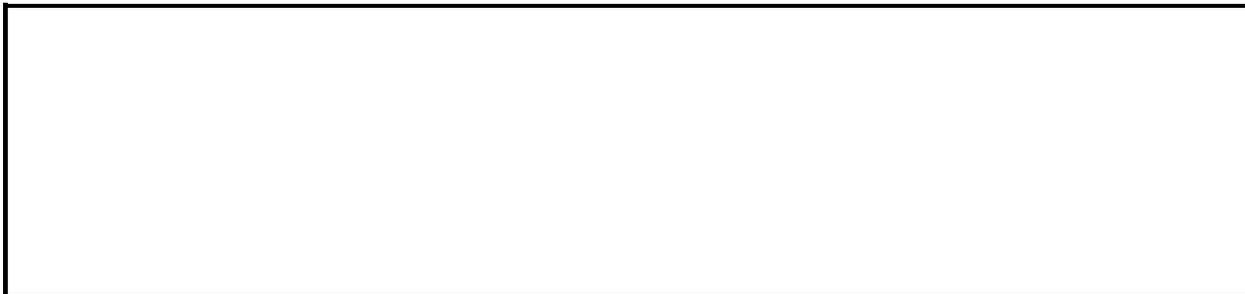
*expression*.**EndConnected**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

If the end of the connector represented by shape three on myDocument is connected to a shape, this example stores the connection site number in the variable `oldEndConnSite`, stores a reference to the connected shape in the object variable `oldEndConnShape`, and then disconnects the end of the connector from the shape.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(3)
    If .Connector Then
        With .ConnectorFormat
            If .EndConnected Then
                oldEndConnSite = .EndConnectionSite
                Set oldEndConnShape = .EndConnectedShape
                .EndDisconnect
            End If
        End With
    End If
End With
```



# EndConnectedShape Property

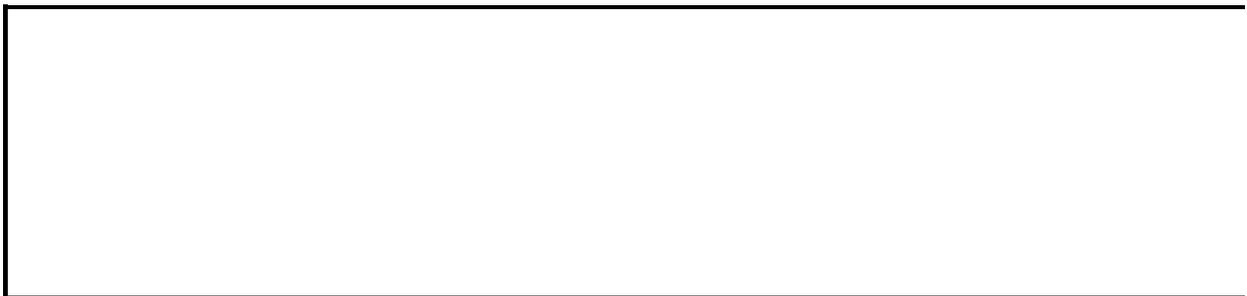
Returns a [Shape](#) object that represents the shape that the end of the specified connector is attached to. Read-only.

**Note** If the end of the specified connector isn't attached to a shape, this property generates an error.

## Example

This example assumes that myDocument already contains two shapes attached by a connector named "Conn1To2." The code adds a rectangle and a connector to myDocument. The end of the new connector will be attached to the same connection site as the end of the connector named "Conn1To2," and the beginning of the new connector will be attached to connection site one on the new rectangle.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes
    Set r3 = .AddShape(msoShapeRectangle, _
        100, 420, 200, 100)
    With .Item("Conn1To2").ConnectorFormat
        endConnSite1 = .EndConnectionSite
        Set endConnShape1 = .EndConnectedShape
    End With
    With .AddConnector(msoConnectorCurve, _
        0, 0, 10, 10).ConnectorFormat
        .BeginConnect r3, 1
        .EndConnect endConnShape1, endConnSite1
    End With
End With
```



# EndConnectionSite Property

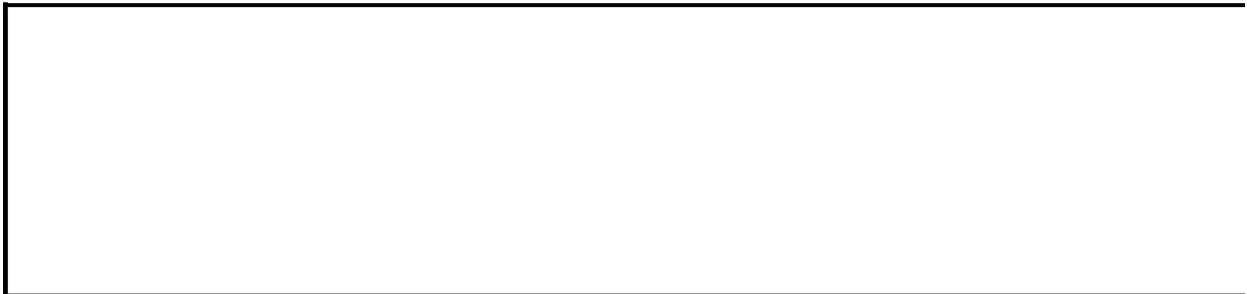
Returns an integer that specifies the connection site that the end of a connector is connected to. Read-only **Long**.

**Note** If the end of the specified connector isn't attached to a shape, this property generates an error.

## Example

This example assumes that myDocument already contains two shapes attached by a connector named "Conn1To2." The code adds a rectangle and a connector to myDocument. The end of the new connector will be attached to the same connection site as the end of the connector named "Conn1To2," and the beginning of the new connector will be attached to connection site one on the new rectangle.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes
    Set r3 = .AddShape(msoShapeRectangle, _
        100, 420, 200, 100)
    With .Item("Conn1To2").ConnectorFormat
        endConnSite1 = .EndConnectionSite
        Set endConnShape1 = .EndConnectedShape
    End With
    With .AddConnector(msoConnectorCurve, _
        0, 0, 10, 10).ConnectorFormat
        .BeginConnect r3, 1
        .EndConnect endConnShape1, endConnSite1
    End With
End With
```



# EndStyle Property

Returns or sets the end style for the error bars. Can be one of the following **XlEndStyleCap** constants: **xlCap** or **xlNoCap**. Read/write **Long**.

## Example

This example sets the end style for the error bars for series one in Chart1. The example should be run on a 2-D line chart that has Y error bars for the first series.

```
Charts("Chart1").SeriesCollection(1).ErrorBars.EndStyle = xlCap
```



# EntireColumn Property

Returns a [Range](#) object that represents the entire column (or columns) that contains the specified range. Read-only.

## Example

This example sets the value of the first cell in the column that contains the active cell. The example must be run from a worksheet.

```
ActiveCell.EntireColumn.Cells(1, 1).Value = 5
```



# EntireRow Property

Returns a [Range](#) object that represents the entire row (or rows) that contains the specified range. Read-only.

## Example

This example sets the value of the first cell in the row that contains the active cell. The example must be run from a worksheet.

```
ActiveCell.EntireRow.Cells(1, 1).Value = 5
```



# EnvelopeVisible Property

**True** if the e-mail composition header and the envelope toolbar are both visible.  
Read/write **Boolean**.

## Example

This example checks to see whether the e-mail composition header and the envelope toolbar are visible in the first workbook. If they are visible, the example then sets the variable strSubject to the text of the e-mail subject line.

```
If Workbooks(1).EnvelopeVisible = True Then  
    strSubject = "Please read: Review immediately"  
End If
```



# ErrorBars Property

Returns an [ErrorBars](#) object that represents the error bars for the series. Read-only.

## Example

This example sets the error bar color for series one in Chart1. The example should be run on a 2-D line chart that has error bars for series one.

```
With Charts("Chart1").SeriesCollection(1)  
    .ErrorBars.Border.ColorIndex = 8  
End With
```



# ErrorCheckingOptions Property

Returns an [ErrorCheckingOptions](#) object, which represents the error checking options for an application.

*expression*.**ErrorCheckingOptions**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

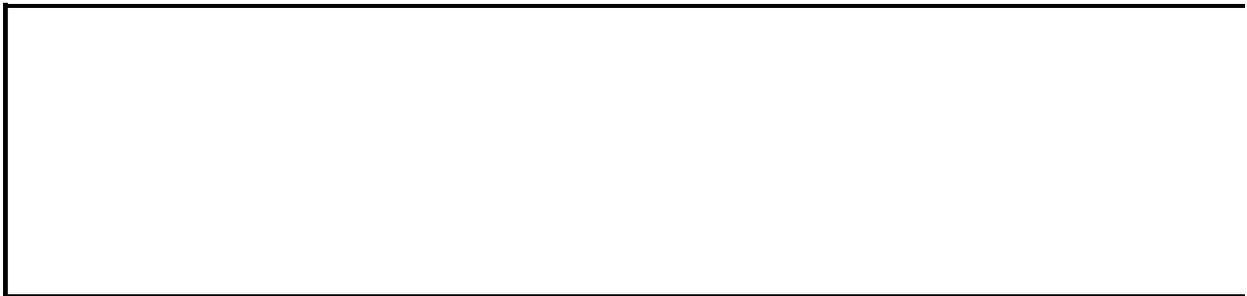
In this example, the **TextDate** property is used in conjunction with the **ErrorCheckingOptions** property. When the user selects a cell containing a two-digit year in the date, the AutoCorrect Options button appears.

```
Sub CheckTextDate()
```

```
    ' Enable Microsoft Excel to identify dates written as text.  
    Application.ErrorCheckingOptions.TextDate = True
```

```
    Range("A1").Formula = "'April 23, 00"
```

```
End Sub
```



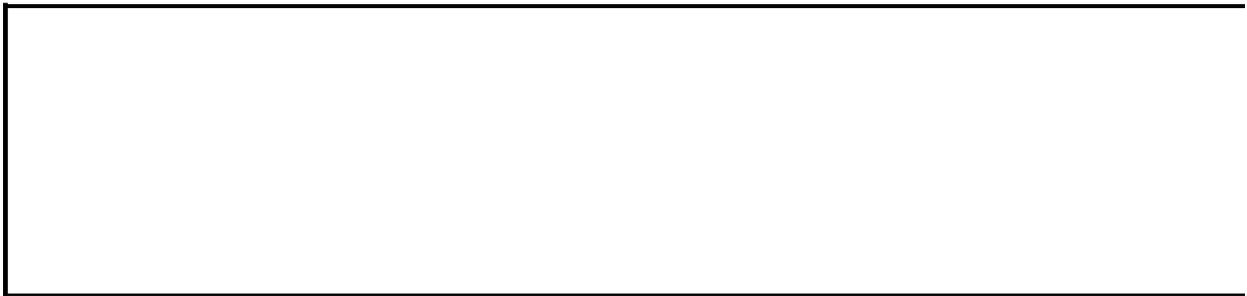
# **ErrorMessage Property**

Returns or sets the data validation error message. Read/write **String**.

## Example

This example adds data validation to cell E5 and specifies both the input and error messages.

```
With Range("e5").Validation
    .Add Type:=xlValidateWholeNumber, _
        AlertStyle:= xlValidAlertStop, _
        Operator:=xlBetween, Formula1:="5", Formula2:="10"
    .InputTitle = "Integers"
    .ErrorTitle = "Integers"
    .InputMessage = "Enter an integer from five to ten"
    .ErrorMessage = "You must enter a number from five to ten"
End With
```



# Errors Property

Allows the user to to access error checking options.

*expression*.**Errors**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Reference the [Errors](#) object to view a list of index values associated with error checking options.

## Example

In this example, a number written as text is placed in cell A1. Microsoft Excel then determines if the number is written as text in cell A1 and notifies the user accordingly.

```
Sub CheckForErrors()  
    Range("A1").Formula = "'12"  
    If Range("A1").Errors.Item(xlNumberAsText).Value = True Then  
        MsgBox "The number is written as text."  
    Else  
        MsgBox "The number is not written as text."  
    End If  
End Sub
```



[Show All](#)

# ErrorString Property

 [ErrorString property as it applies to the \*\*PivotTable\*\* object.](#)

Returns or sets the string displayed in cells that contain errors when the **DisplayErrorString** property is **True**. The default value is an empty string ("").  
Read/write **String**.

*expression*.**ErrorString**

*expression* Required. An expression that returns one of the above objects.

 [ErrorString property as it applies to the \*\*ODBCError\*\* and \*\*OLEDBError\*\* objects.](#)

Returns the ODBC error string. Read-only **String**.

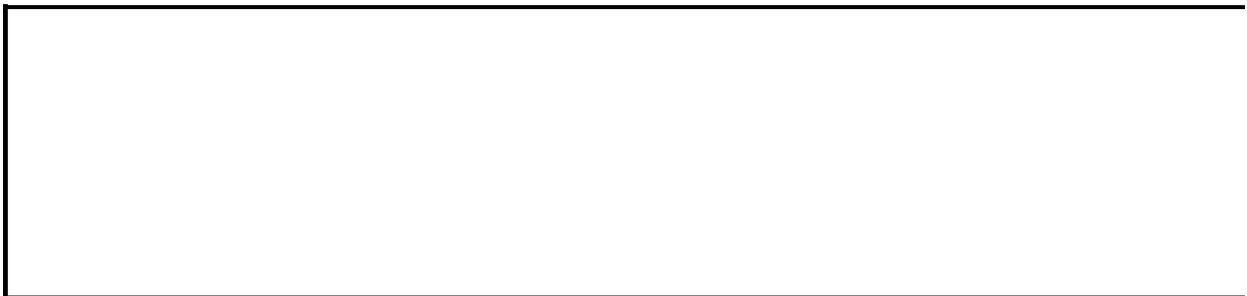
*expression*.**ErrorString**

*expression* Required. An expression that returns one of the above objects.

## Example

This example displays a hyphen in cells in the specified PivotTable report that contain errors.

```
With Worksheets(1).PivotTables("Pivot1")  
    .ErrorString = "-"  
    .DisplayErrorString = True  
End With
```



# ErrorTitle Property

Returns or sets the title of the data-validation error dialog box. Read/write **String**.

## Example

This example adds data validation to cell E5.

```
With Range("e5").Validation
    .Add xlValidateWholeNumber, _
        xlValidAlertInformation, xlBetween, "5", "10"
    .InputTitle = "Integers"
    .ErrorTitle = "Integers"
    .InputMessage = "Enter an integer from five to ten"
    .ErrorMessage = "You must enter a number from five to ten"
End With
```



# EvaluateToError Property

When set to **True** (default), Microsoft Excel identifies, with an AutoCorrect Options button, selected cells that contain formulas evaluating to an error. **False** disables error checking for cells that evaluate to an error value. Read/write **Boolean**.

*expression*.**EvaluateToError**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In the following example, the AutoCorrect Options button appears for cell A3, which contains a divide-by-zero error.

```
Sub CheckEvaluationError()  
    ' Simulate a divide-by-zero error.  
    Application.ErrorCheckingOptions.EvaluateToError = True  
    Range("A1").Value = 1  
    Range("A2").Value = 0  
    Range("A3").Formula = "=A1/A2"  
End Sub
```



# Excel4IntlMacroSheets Property

Returns a [Sheets](#) collection that represents all the Microsoft Excel 4.0 international macro sheets in the specified workbook. Read-only.

For information about returning a single member of a collection, see [Returning an Object from a Collection](#).

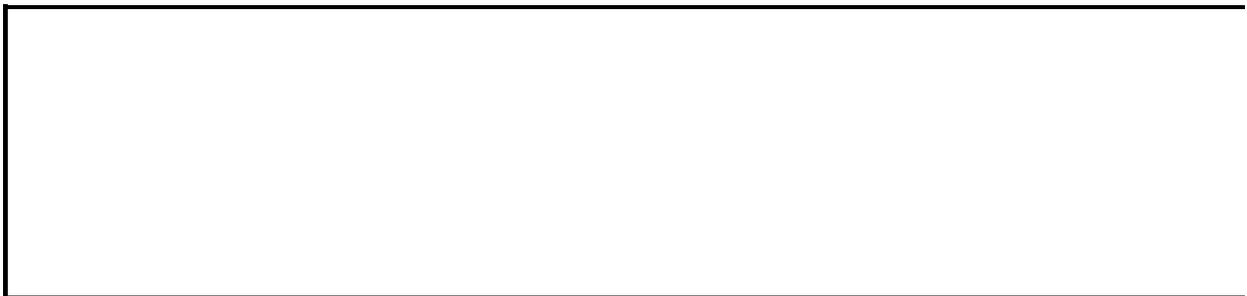
## Remarks

Using this property with the **Application** object or without an object qualifier is equivalent to using `ActiveWorkbook.Excel4Int1MacroSheets`.

## Example

This example displays the number of Microsoft Excel 4.0 international macro sheets in the active workbook.

```
MsgBox "There are " & _  
    ActiveWorkbook.Excel4IntlMacroSheets.Count & _  
    " Microsoft Excel 4.0 international macro sheets" & _  
    " in this workbook."
```



# Excel4MacroSheets Property

Returns a [Sheets](#) collection that represents all the Microsoft Excel 4.0 macro sheets in the specified workbook. Read-only.

For information about returning a single member of a collection, see [Returning an Object from a Collection](#).

## Remarks

Using this property with the **Application** object or without an object qualifier is equivalent to using `ActiveWorkbook.Excel4MacroSheets`.

## Example

This example displays the number of Microsoft Excel 4.0 macro sheets in the active workbook.

```
MsgBox "There are " & ActiveWorkbook.Excel14MacroSheets.Count & _  
      " Microsoft Excel 4.0 macro sheets in this workbook."
```



# ExpandHelp Property

Sets or returns a **Boolean** value that represents whether the specified smart document Help text control is expanded or collapsed in the **Document Actions** task pane. **True** indicates that the control is expanded. **False** indicates that the control is collapsed.

*expression*.**ExpandHelp**

*expression* Required. An expression that returns a [SmartTagAction](#) object.

## Remarks

For more information on smart documents, refer to the Smart Document Software Development Kit on the [Microsoft Developer Network \(MSDN\)](#) Web site.

---

---

---

# Explosion Property

Returns or sets the explosion value for a pie-chart or doughnut-chart slice. Returns 0 (zero) if there's no explosion (the tip of the slice is in the center of the pie). Read/write **Long**.

## Example

This example sets the explosion value for point two in Chart1. The example should be run on a pie chart.

```
Charts("Chart1").SeriesCollection(1).Points(2).Explosion = 20
```



# ExtendList Property

**True** if Microsoft Excel automatically extends formatting and formulas to new data that is added to a list. Read/write **Boolean**.

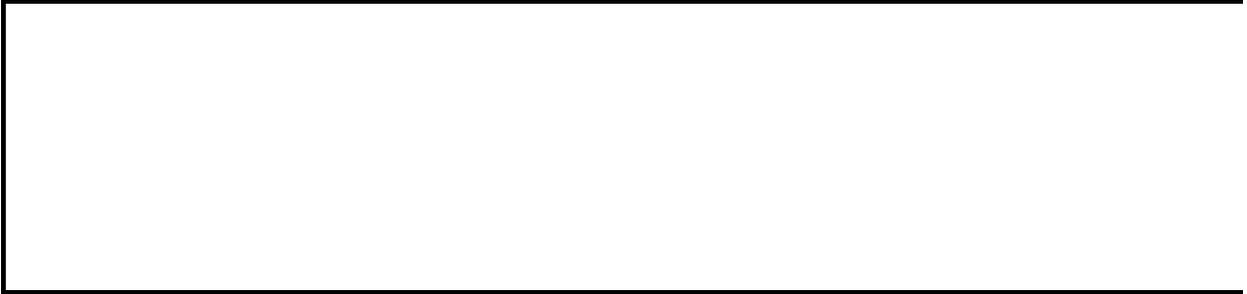
## Remarks

To be extended, formats and formulas must appear in at least three of the five list rows or columns preceding the new row or column, and you must add the data to the bottom or to the right-hand side of the list.

## Example

This example sets Excel to not apply formatting and formulas to data subsequently added to an existing list.

```
Application.ExtendList = False
```



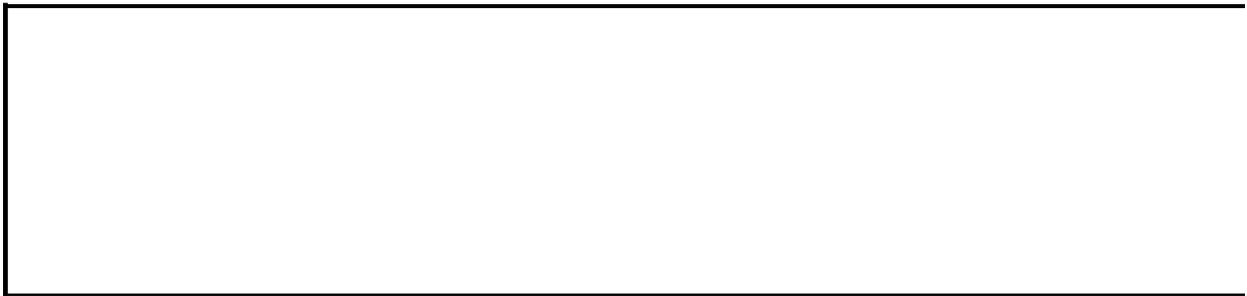
# Extent Property

Returns the type of the specified page break: full-screen or only within a print area. Can be either of the following **XlPageBreakExtent** constants: **xlPageBreakFull** or **xlPageBreakPartial**. Read-only **Long**.

## Example

This example displays the total number of full-screen and print-area horizontal page breaks.

```
For Each pb in Worksheets(1).HPageBreaks
    If pb.Extent = xlPageBreakFull Then
        cFull = cFull + 1
    Else
        cPartial = cPartial + 1
    End If
Next
MsgBox cFull & " full-screen page breaks, " & cPartial & _
    " print-area page breaks"
```



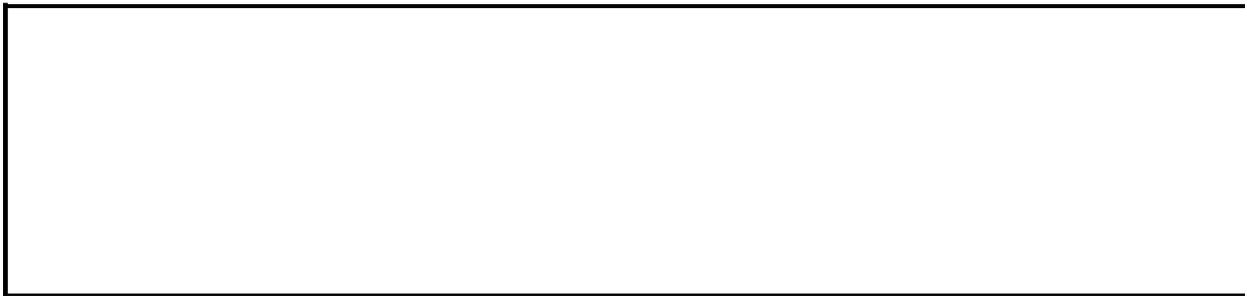
# ExtrusionColor Property

Returns a [ColorFormat](#) object that represents the color of the shape's extrusion.  
Read-only.

## Example

This example adds an oval to myDocument and then specifies that the oval be extruded to a depth of 50 points and that the extrusion be purple.

```
Set myDocument = Worksheets(1)
Set myShape = myDocument.Shapes.AddShape(msoShapeOval, _
    90, 90, 90, 40)
With myShape.ThreeD
    .Visible = True
    .Depth = 50
    .ExtrusionColor.RGB = RGB(255, 100, 255)
    ' RGB value for purple
End With
```



[Show All](#)

# ExtrusionColorType Property

Returns or sets a value that indicates whether the extrusion color is based on the extruded shape's fill (the front face of the extrusion) and automatically changes when the shape's fill changes, or whether the extrusion color is independent of the shape's fill. Read/write [MsoExtrusionColorType](#).

MsoExtrusionColorType can be one of these MsoExtrusionColorType constants.

**msoExtrusionColorAutomatic.** Extrusion color based on shape fill.

**msoExtrusionColorTypeMixed**

**msoExtrusionColorCustom.** Extrusion color independent of shape fill.

*expression*.**ExtrusionColorType**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

If shape one on myDocument has an automatic extrusion color, this example gives the extrusion a custom yellow color.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(1).ThreeD
    If .ExtrusionColorType = msoExtrusionColorAutomatic Then
        .ExtrusionColor.RGB = RGB(240, 235, 16)
    End If
End With
```



[Show All](#)

# FeatureInstall Property

Returns or sets a value (constant) that specifies how Microsoft Excel handles calls to methods and properties that require features that aren't yet installed. Can be one of the **MsoFeatureInstall** constants listed in the following table.

Read/write [MsoFeatureInstall](#).

MsoFeatureInstall can be one of these MsoFeatureInstall constants.

**msoFeatureInstallNone.** Generates a generic Automation error at run time when uninstalled features are called. This is the default constant

**msoFeatureInstallOnDemand.** Prompts the user to install new features

**msoFeatureInstallOnDemandWithUI.** Displays a progress meter during installation; doesn't prompt the user to install new features.

*expression*.**FeatureInstall**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

You can use the **msoFeatureInstallOnDemandWithUI** constant to prevent users from thinking that the application isn't responding while a feature is being installed. Use the **msoFeatureInstallNone** constant if you want the developer to be the only one who can install features.

If you have the [DisplayAlerts](#) property set to **False**, users won't be prompted to install new features even if the **FeatureInstall** property is set to **msoFeatureInstallOnDemand**. If the **DisplayAlerts** property is set to **True**, an installation progress meter will appear if the **FeatureInstall** property is set to **msoFeatureInstallOnDemand**.

## Example

This example activates a new instance of Microsoft Word and checks the value of the **FeatureInstall** property. Be sure to set a reference to the Microsoft Word object library. If the **FeatureInstall** property is set to **msoFeatureInstallNone**, the code displays a message box that asks the user whether they want to change the property setting. If the user responds Yes, the property is set to **msoFeatureInstallOnDemand**.

```
Dim WordApp As New Word.Application, Reply As Integer
Application.ActivateMicrosoftApp xlMicrosoftWord With WordApp
  If .FeatureInstall = msoFeatureInstallNone Then
    Reply = MsgBox("Uninstalled features for this " _
      & "application " & vbCrLf _
      & "may cause a run-time error when called." & vbCrLf _
      & vbCrLf _
      & "Would you like to change this setting" & vbCrLf _
      & "to automatically install missing features?" _
      , 52, "Feature Install Setting")
    If Reply = 6 Then
      .FeatureInstall = msoFeatureInstallOnDemand
    End If
  End If
End With
```



# FetchRowOverflow Property

**True** if the number of rows returned by the last use of the **Refresh** method is greater than the number of rows available on the worksheet. Read-only **Boolean**.

## Example

This example refreshes query table one. If the number of rows returned by the query exceeds the number of rows available on the worksheet, an error message is displayed.

```
With Worksheets(1).QueryTables(1)
    .Refresh
    If .FetchedRowOverflow Then
        MsgBox "Query too large: please redefine."
    End If
End With
```



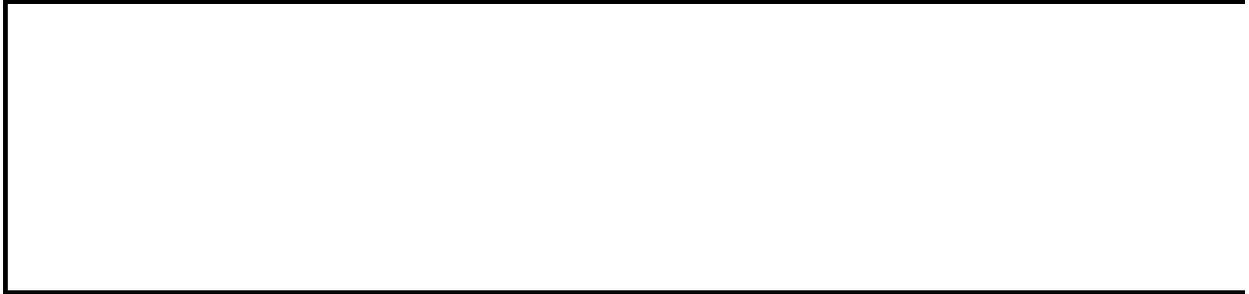
# FieldNames Property

**True** if field names from the data source appear as column headings for the returned data. The default value is **True**. Read/write **Boolean**.

## Example

This example sets query table one so that the field names don't appear in it.

```
Worksheets(1).QueryTables(1).FieldNames = False
```



# FileConverters Property

Returns information about installed file converters. Returns **Null** if there are no converters installed. Read-only **Variant**.

*expression*.**FileConverters**(*Index1*, *Index2*)

*expression* Required. An expression that returns an **Application** object.

**Index1** Optional **Variant**. The long name of the converter, including the file-type search string in Windows (for example, "Lotus 1-2-3 Files (\*.wk\*)").

**Index2** Optional **Variant**. The path of the converter DLL or code resource.

## Remarks

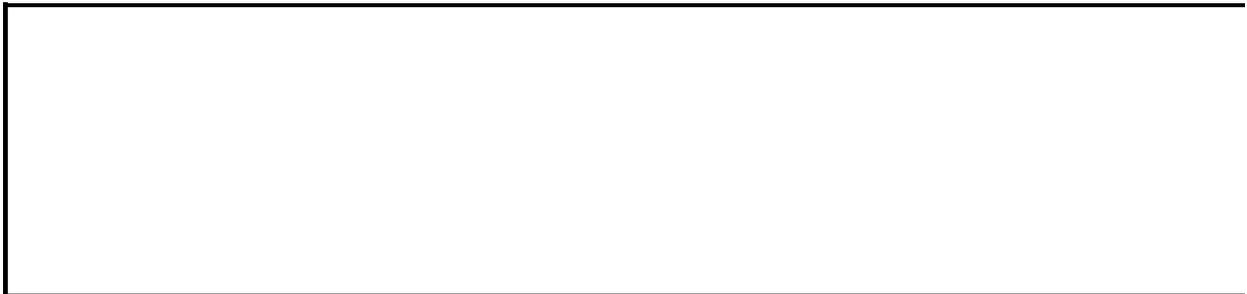
If you don't specify the index arguments, this property returns an array that containing information about all the installed file converters. Each row in the array contains information about a single file converter, as shown in the following table.

<b>Column</b>	<b>Contents</b>
1	The long name of the converter
2	The path of the converter DLL or code resource
3	The file-extension search string

## Example

This example displays a message if the Multiplan file converter is installed.

```
installedCvts = Application.FileConverters
foundMultiplan = False
If Not IsNull(installedCvts) Then
    For arrayRow = 1 To UBound(installedCvts, 1)
        If installedCvts(arrayRow, 1) Like "*Multiplan*" Then
            foundMultiplan = True
            Exit For
        End If
    Next arrayRow
End If
If foundMultiplan = True Then
    MsgBox "Multiplan converter is installed"
Else
    MsgBox "Multiplan converter is not installed"
End If
```



[Show All](#)

# FileDialog Property

Returns a [FileDialog](#) object representing an instance of the file dialog.

*expression*.**FileDialog**(*fileDialogType*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*fileDialogType* Required [MsoFileDialogType](#). The type of file dialog.

MsoFileDialogType can be one of these MsoFileDialogType constants.

**msoFileDialogFilePicker** Allows user to select a file.

**msoFileDialogFolderPicker** Allows user to select a folder.

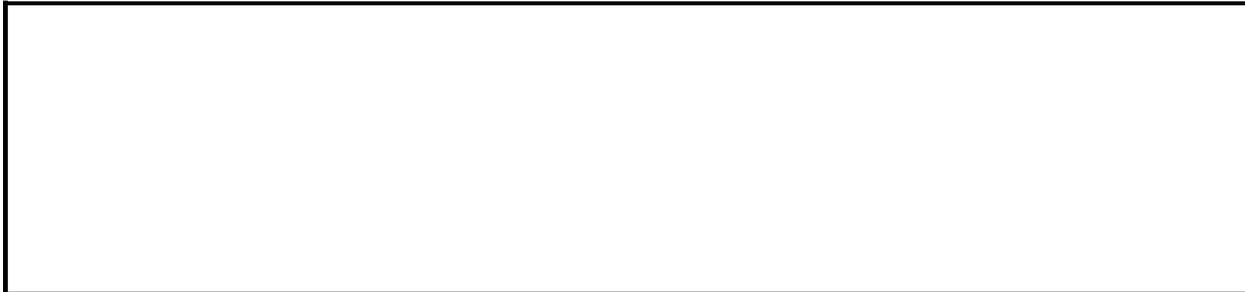
**msoFileDialogOpen** Allows user to open a file.

**msoFileDialogSaveAs** Allows user to save a file.

## Example

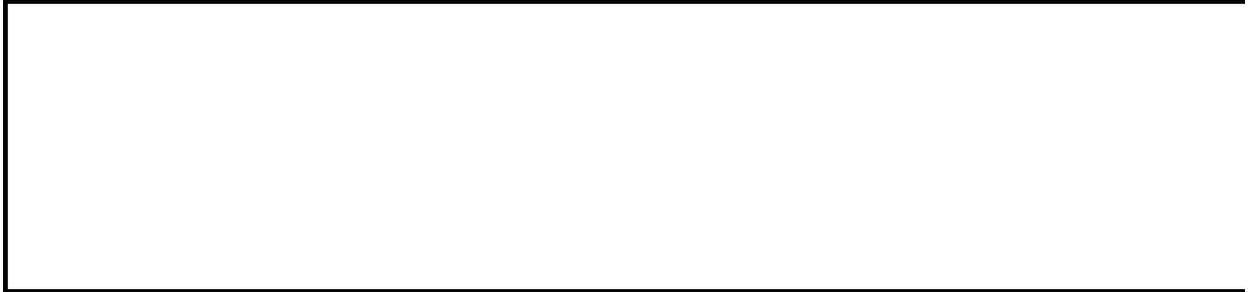
In this example, Microsoft Excel opens the file dialog allowing the user to select one or more files. Once these files are selected, Excel displays the path for each file in a separate message.

```
Sub UseFileDialogOpen()  
  
    Dim lngCount As Long  
  
    ' Open the file dialog  
    With Application.FileDialog(msoFileDialogOpen)  
        .AllowMultiSelect = True  
        .Show  
  
        ' Display paths of each file selected  
        For lngCount = 1 To .SelectedItems.Count  
            MsgBox .SelectedItems(lngCount)  
        Next lngCount  
  
    End With  
  
End Sub
```



# FileFind Property

You have requested Help for a Visual Basic keyword used only on the Macintosh. For information about this keyword, consult the language reference Help included with Microsoft Office Macintosh Edition.



[Show All](#)

# FileFormat Property

Returns the file format and/or type of the workbook. Read-only [XlFileFormat](#).

XlFileFormat can be one of these XlFileFormat constants.

**xlCSV**

**xlCSVMSDOS**

**xlCurrentPlatformText**

**xlDBF3**

**xlDIF**

**xlExcel2FarEast**

**xlExcel4**

**xlAddIn**

**xlCSVMac**

**xlCSVWindows**

**xlDBF2**

**xlDBF4**

**xlExcel2**

**xlExcel3**

**xlExcel4Workbook**

**xlExcel5**

**xlExcel7**

**xlExcel9795**

**xlHtml**

**xlIntlAddIn**

**xlIntlMacro**

**xlSYLK**

**xlTemplate**

**xlTextMac**

**xlTextMSDOS**

**xlTextPrinter**

**xlTextWindows**

**xlUnicodeText**  
**xlWebArchive**  
**xlWJ2WD1**  
**xlWJ3**  
**xlWJ3FJ3**  
**xlWK1**  
**xlWK1ALL**  
**xlWK1FMT**  
**xlWK3**  
**xlWK3FM3**  
**xlWK4**  
**xlWKS**  
**xlWorkbookNormal**  
**xlWorks2FarEast**  
**xlWQ1**  
**xlXMLSpreadsheet**

*expression*.**FileFormat**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Some of these constants may not be available to you, depending on the language support (U.S. English, for example) that you've selected or installed.

## Example

This example saves the active workbook in Normal file format if its current file format is WK3.

```
If ActiveWorkbook.FileFormat = xlWK3 Then  
    ActiveWorkbook.SaveAs fileFormat:=xlNormal  
End If
```



# FileSearch Property

Returns a [FileSearch](#) object for use with file searches. This property is available only in Microsoft Windows.

## Example

This example creates a **FoundFiles** object that represents all the Microsoft Excel workbooks in the My Documents folder.

```
With Application.FileSearch  
    .LookIn = "c:\my documents"  
    .FileType = msoFileTypeExcelWorkbooks  
    .Execute  
End With
```



# Fill Property

Returns a [FillFormat](#) object that contains fill formatting properties for the specified chart or shape. Read-only.

## Example

This example adds a rectangle to myDocument and then sets the foreground color, background color, and gradient for the rectangle's fill.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddShape(msoShapeRectangle, _
    90, 90, 90, 50).Fill
    .ForeColor.RGB = RGB(128, 0, 0)
    .BackColor.RGB = RGB(170, 170, 170)
    .TwoColorGradient msoGradientHorizontal, 1
End With
```



# FillAdjacentFormulas Property

**True** if formulas to the right of the specified query table are automatically updated whenever the query table is refreshed. Read/write **Boolean**.

## Example

This example sets query table one so that formulas to the right of it are automatically updated whenever the query table is refreshed.

```
Sheets("sheet1").QueryTables(1).FillAdjacentFormulas = True
```



# FilterMode Property

**True** if the worksheet is in filter mode. Read-only **Boolean**.

## Remarks

This property is **True** if the worksheet contains a filtered list in which there are hidden rows.

## Example

This example displays the filter status of Sheet1 in a message box.

```
If Worksheets("Sheet1").FilterMode = True Then  
    MsgBox "Filter mode is on"  
Else  
    MsgBox "Filter mode is off"  
End If
```



# Filters Property

Returns a [Filters](#) collection that represents all the filters in an autofiltered range.  
Read-only.

## Example

The following example sets a variable to the value of the **Criteria1** property of the filter for the first column in the filtered range on the Crew worksheet.

```
With Worksheets("Crew")
    If .AutoFilterMode Then
        With .AutoFilter.Filters(1)
            If .On Then c1 = .Criteria1
        End With
    End If
End With
```



# FindFormat Property

Sets or returns the search criteria for the type of cell formats to find.

*expression*.**FindFormat**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, the search criteria is set to look for Arial, Regular, Size 10 font cells and the user is notified.

```
Sub UseFindFormat()  
    ' Establish search criteria.  
    With Application.FindFormat.Font  
        .Name = "Arial"  
        .FontStyle = "Regular"  
        .Size = 10  
    End With  
  
    ' Notify user.  
    With Application.FindFormat.Font  
        MsgBox .Name & "-" & .FontStyle & "-" & .Size & _  
            " font is what the search criteria is set to."  
    End With  
  
End Sub
```



# FirstChild Property

Returns a [DiagramNode](#) object that represents the first child node of a parent node.

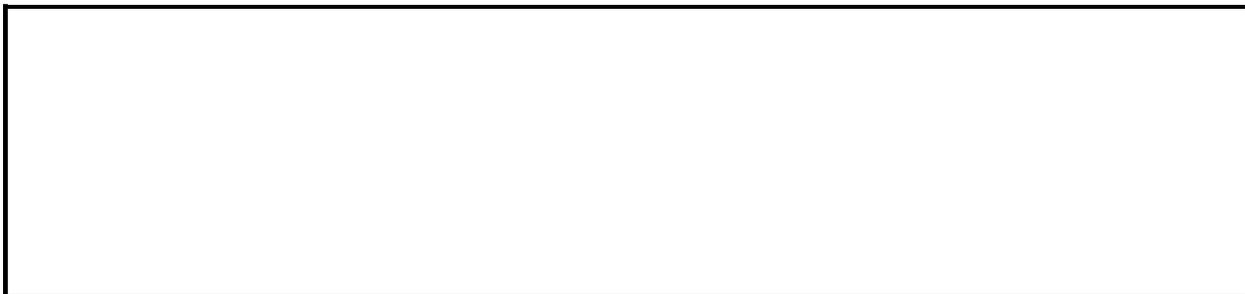
*expression*.**FirstChild**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example adds an organization chart diagram to the current worksheet, adds three nodes, and assigns the first and last child nodes to variables.

```
Sub FirstChild()  
  
    Dim shpDiagram As Shape  
    Dim dgnRoot As DiagramNode  
    Dim dgnFirstChild As DiagramNode  
    Dim dgnLastChild As DiagramNode  
    Dim intCount As Integer  
  
    'Add organizational chart diagram to the current document  
    Set shpDiagram = ActiveSheet.Shapes.AddDiagram( _  
        Type:=msoDiagramOrgChart, Left:=10, _  
        Top:=15, Width:=400, Height:=475)  
  
    'Add the first node to the diagram  
    Set dgnRoot = shpDiagram.DiagramNode.Children.AddNode  
  
    'Add three child nodes  
    For intCount = 1 To 3  
        dgnRoot.Children.AddNode  
    Next intCount  
  
    'Assign the first and last child nodes to variables  
    Set dgnFirstChild = dgnRoot.Children.FirstChild  
    Set dgnLastChild = dgnRoot.Children.LastChild  
  
End Sub
```



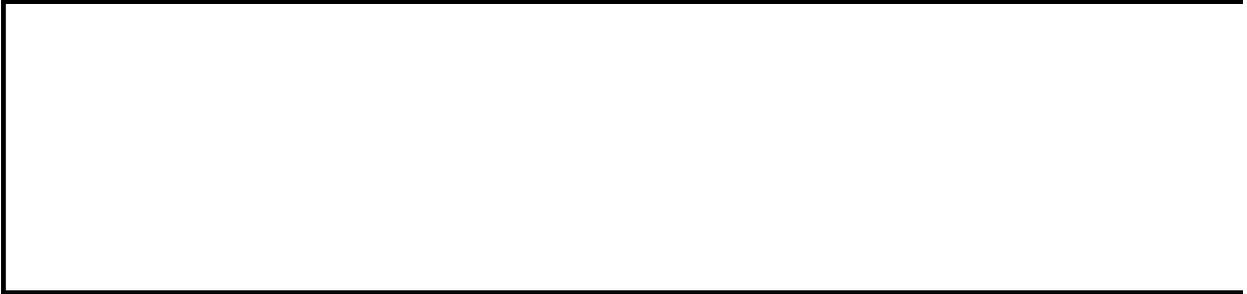
# FirstPageNumber Property

Returns or sets the first page number that will be used when this sheet is printed. If **xlAutomatic**, Microsoft Excel chooses the first page number. The default is **xlAutomatic**. Read/write **Long**.

## Example

This example sets the first page number of Sheet1 to 100.

```
Worksheets("Sheet1").PageSetup.FirstPageNumber = 100
```



# FirstSliceAngle Property

Returns or sets the angle of the first pie-chart or doughnut-chart slice, in degrees (clockwise from vertical). Applies only to pie, 3-D pie, and doughnut charts.  
Read/write **Long**.

## Example

This example sets the angle for the first slice in chart group one in Chart1. The example should be run on a 2-D pie chart.

```
Charts("Chart1").ChartGroups(1).FirstSliceAngle = 15
```



# FitToPagesTall Property

Returns or sets the number of pages tall the worksheet will be scaled to when it's printed. Applies only to worksheets. Read/write **Variant**.

## Remarks

If this property is **False**, Microsoft Excel scales the worksheet according to the [FitToPagesWide](#) property.

If the [Zoom](#) property is **True**, the **FitToPagesTall** property is ignored.

## Example

This example causes Microsoft Excel to print Sheet1 exactly one page tall and wide.

```
With Worksheets("Sheet1").PageSetup  
    .Zoom = False  
    .FitToPagesTall = 1  
    .FitToPagesWide = 1  
End With
```



# FitToPagesWide Property

Returns or sets the number of pages wide the worksheet will be scaled to when it's printed. Applies only to worksheets. Read/write **Variant**.

## Remarks

If this property is **False**, Microsoft Excel scales the worksheet according to the [FitToPagesTall](#) property.

If the [Zoom](#) property is **True**, the **FitToPagesWide** property is ignored.

## Example

This example causes Microsoft Excel to print Sheet1 exactly one page wide and tall.

```
With Worksheets("Sheet1").PageSetup
    .Zoom = False
    .FitToPagesTall = 1
    .FitToPagesWide = 1
End With
```



# FixedDecimal Property

All data entered after this property is set to **True** will be formatted with the number of fixed decimal places set by the [FixedDecimalPlaces](#) property.  
Read/write **Boolean**.

## Example

This example sets the **FixedDecimal** property to **True** and then sets the **FixedDecimalPlaces** property to 4. Entering "30000" after running this example produces "3" on the worksheet, and entering "12500" produces "1.25."

```
Application.FixedDecimal = True  
Application.FixedDecimalPlaces = 4
```



# FixedDecimalPlaces Property

Returns or sets the number of fixed decimal places used when the [FixedDecimal](#) property is set to **True**. Read/write **Long**.

## Example

This example sets the **FixedDecimal** property to **True** and then sets the **FixedDecimalPlaces** property to 4. Entering "30000" after running this example produces "3" on the worksheet, and entering "12500" produces "1.25."

```
Application.FixedDecimal = True  
Application.FixedDecimalPlaces = 4
```



# Floor Property

Returns a [Floor](#) object that represents the floor of the 3-D chart. Read-only.

For information about using the **Floor** worksheet function in Visual Basic, see [Using Worksheet Functions in Visual Basic](#).

## Example

This example sets the floor color of Chart1 to blue. The example should be run on a 3-D chart (the **Floor** property fails on 2-D charts).

```
Charts("Chart1").Floor.Interior.ColorIndex = 5
```



# FolderSuffix Property

Returns the folder suffix that Microsoft Excel uses when you save a document as a Web page, use long file names, and choose to save supporting files in a separate folder (that is, if the [UseLongFileNames](#) and [OrganizeInFolder](#) properties are set to **True**). Read-only **String**.

*expression*.**FolderSuffix**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Newly created documents use the suffix returned by the **FolderSuffix** property of the **DefaultWebOptions** object. The value of the **FolderSuffix** property of the **WebOptions** object may differ from that of the **DefaultWebOptions** object if the document was previously edited in a different language version of Microsoft Excel. You can use the [UseDefaultFolderSuffix](#) method to change the suffix to the language you are currently using in Microsoft Office.

By default, the name of the supporting folder is the name of the Web page plus an underscore (\_), a period (.), or a hyphen (-) and the word "files" (appearing in the language of the version of Excel in which the file was saved as a Web page). For example, suppose that you use the Dutch language version of Excel to save a file called "Page1" as a Web page. The default name of the supporting folder is Page1\_bestanden.

The following table lists each language version of Office, and gives its corresponding **LanguageID** property value and folder suffix. For the languages that are not listed in the table, the suffix ".files" is used.

Language	LanguageID	Folder suffix
Arabic	1025	.files
Basque	1069	_fitxategiak
Brazilian	1046	_arquivos
Bulgarian	1026	.files
Catalan	1027	_fitxers
Chinese - Simplified	2052	.files
Chinese - Traditional	1028	.files
Croatian	1050	_datoteke
Czech	1029	_soubory
Danish	1030	-filer
Dutch	1043	_bestanden
English	1033	_files
Estonian	1061	_failid
Finnish	1035	_tiedostot

French	1036	_fichiers
German	1031	-Dateien
Greek	1032	.files
Hebrew	1037	.files
Hungarian	1038	_elemei
Italian	1040	-file
Japanese	1041	.files
Korean	1042	.files
Latvian	1062	_fails
Lithuanian	1063	_bylos
Norwegian	1044	-filer
Polish	1045	_pliki
Portuguese	2070	_ficheiros
Romanian	1048	.files
Russian	1049	.files
Serbian (Cyrillic)	3098	.files
Serbian (Latin)	2074	_fajlovi
Slovakian	1051	.files
Slovenian	1060	_datoteke
Spanish	3082	_archivos
Swedish	1053	-filer
Thai	1054	.files
Turkish	1055	_dosyalar
Ukranian	1058	.files
Vietnamese	1066	.files

## Example

This example returns the folder suffix used by the first workbook. The suffix is returned in the string variable `strFolderSuffix`.

```
strFolderSuffix = Workbooks(1).WebOptions.FolderSuffix
```



[Show All](#)

# Font Property

[Font property as it applies to the \*\*CellFormat\*\* object.](#)

Returns a **Font** object, allowing the user to set or return the search criteria based on the cell's font format.

*expression*.**Font**

*expression* Required. An expression that returns one of the above objects.

[Font property as it applies to all other objects in the \*\*Applies To\*\* list.](#)

Returns a **Font** object that represents the font of the specified object.

*expression*.**Font**

*expression* Required. An expression that returns one of the above objects.

## Example

 [As it applies to the \*\*CellFormat\*\* object.](#)

This example sets the search criteria to identify cells that contain red font, creates a cell with this condition, finds this cell, and notifies the user.

```
Sub SearchCellFormat()  
  
    ' Set the search criteria for the font of the cell format.  
    Application.FindFormat.Font.ColorIndex = 3  
  
    ' Set the color index of the font for cell A5 to red.  
    Range("A5").Font.ColorIndex = 3  
    Range("A5").Formula = "Red font"  
    Range("A1").Select  
    MsgBox "Cell A5 has red font"  
  
    ' Find the cells based on the search criteria.  
    Cells.Find(What:="", After:=ActiveCell, LookIn:=xlFormulas, Look  
        xlPart, SearchOrder:=xlByRows, SearchDirection:=xlNext, Matc  
        , SearchFormat:=True).Activate  
  
    MsgBox "Microsoft Excel has found this cell matching the search  
  
End Sub
```

 [As it applies to all other objects in the \*\*Applies To\*\* list.](#)

This example determines the if the font name for cell A1 is Arial and notifies the user.

```
Sub CheckFont()  
  
    Range("A1").Select  
  
    ' Determine if the font name for selected cell is Arial.  
    If Range("A1").Font.Name = "Arial" Then  
        MsgBox "The font name for this cell is 'Arial'"  
    Else  
        MsgBox "The font name for this cell is not 'Arial'"  
    End If
```

End Sub



[Show All](#)

# FontBold Property

**True** if the font in the specified WordArt is bold. Read/write [MsoTriState](#).

MsoTriState can be one of these MsoTriState constants.

**msoCTrue**

**msoFalse**

**msoTriStateMixed**

**msoTriStateToggle**

**msoTrue** The specified WordArt is bold.

*expression*.**FontBold**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example sets the font to bold for shape three on myDocument if the shape is WordArt.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(3)
    If .Type = msoTextEffect Then
        .TextEffect.FontBold = msoTrue
    End If
End With
```



[Show All](#)

# FontItalic Property

Returns **msoTrue** if the font in the specified WordArt is italic. Read/write [MsoTriState](#).

MsoTriState can be one of these MsoTriState constants.

**msoCTrue** Does not apply to this property.

**msoFalse** The specified WordArt is not italic.

**msoTriStateMixed** Does not apply to this property.

**msoTriStateToggle** Does not apply to this property.

**msoTrue** The specified WordArt is italic.

*expression*.**FontBold**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example sets the font to italic for the shape named "WordArt 4" in myDocument.

```
Set myDocument = Worksheets(1)  
myDocument.Shapes("WordArt 4").TextEffect.FontItalic = msoTrue
```



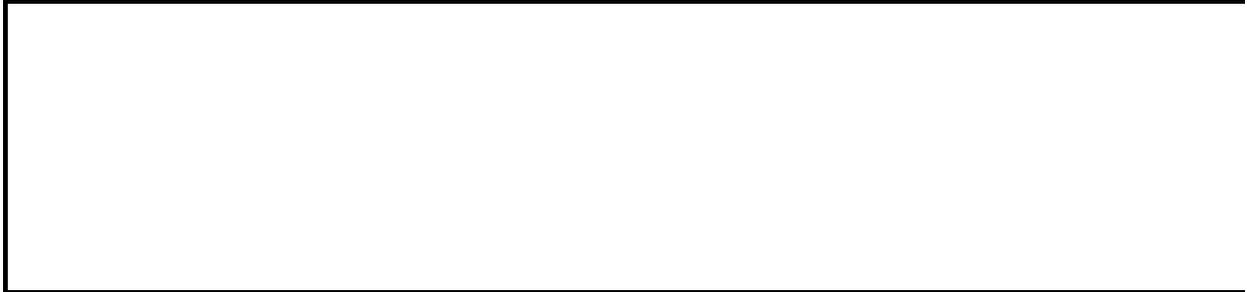
# FontName Property

Returns or sets the name of the font in the specified WordArt. Read/write **String**.

## Example

This example sets the font name to "Courier New" for shape three on myDocument if the shape is WordArt.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(3)
    If .Type = msoTextEffect Then
        .TextEffect.FontName = "Courier New"
    End If
End With
```



# Fonts Property

Returns the [WebPageFonts](#) collection representing the set of fonts Microsoft Excel uses when you open a Web page in Excel and there is either no font information specified in the Web page, or the current default font can't display the character set in the Web page. Read-only.

## Example

This example sets the default fixed-width font for the English/Western European/Other Latin Script character set to Courier New, 14 points.

```
With Application.DefaultWebOptions _  
    .Fonts(msoCharacterSetEnglishWesternEuropeanOtherLatinScript)  
        .FixedWidthFont = "Courier New"  
        .FixedWidthFontSize = 14  
End With
```



# FontSize Property

Returns or sets the font size for the specified WordArt, in points. Read/write **Single**.

## Example

This example sets the font size to 16 points for the shape named "WordArt 4" in myDocument.

```
Set myDocument = Worksheets(1)  
myDocument.Shapes("WordArt 4").TextEffect.FontSize = 16
```



# FontStyle Property

Returns or sets the font style. Read/write **String**.

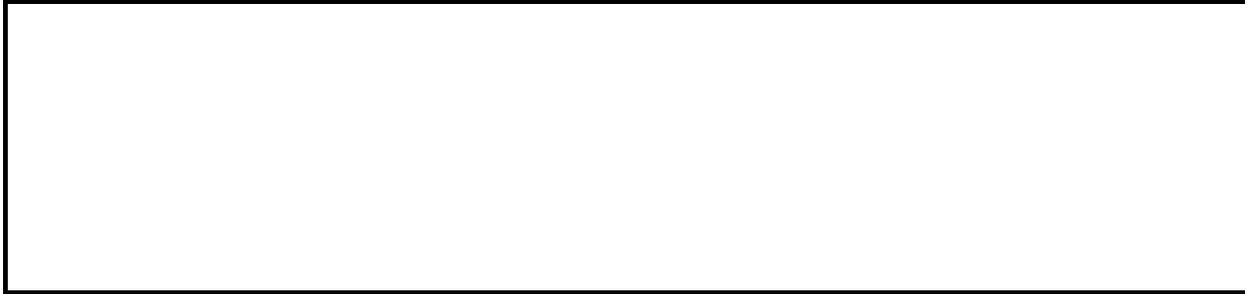
## Remarks

Changing this property may affect other **Font** properties (such as **Bold** and **Italic**).

## Example

This example sets the font style for cell A1 on Sheet1 to bold and italic.

```
Worksheets("Sheet1").Range("A1").Font.FontStyle = "Bold Italic"
```



[Show All](#)

# FooterMargin Property

Returns or sets the distance from the bottom of the page to the footer, in [points](#).  
Read/write **Double**.

## Example

This example sets the footer margin of Sheet1 to 0.5 inch.

```
Worksheets("Sheet1").PageSetup.FooterMargin = _  
    Application.InchesToPoints(0.5)
```



[Show All](#)

# ForeColor Property

 [ForeColor property as it applies to the \*\*ChartFillFormat\*\* object.](#)

Returns a [ChartColorFormat](#) object that represents the specified foreground fill or solid color. Read-only **ChartColorFormat** object.

*expression*.**ForeColor**

*expression* Required. An expression that returns one of the above objects.

 [ForeColor property as it applies to the \*\*FillFormat\*\*, \*\*LineFormat\*\*, and \*\*ShadowFormat\*\* objects.](#)

Returns a [ColorFormat](#) object that represents the specified foreground fill or solid color. Read/write **ColorFormat** object.

*expression*.**ForeColor**

*expression* Required. An expression that returns one of the above objects.

## Example

This example sets the foreground color, background color, and gradient for the chart area fill on chart one.

```
With Charts(1).ChartArea.Fill  
    .Visible = True  
    .ForeColor.SchemeColor = 15  
    .BackColor.SchemeColor = 17  
    .TwoColorGradient msoGradientHorizontal, 1  
End With
```



# FormatConditions Property

Returns a [FormatConditions](#) collection that represents all the conditional formats for the specified range. Read-only.

For more information about returning an individual member of a collection, see [Returning an Object from a Collection](#).

## Example

This example modifies an existing conditional format for cells E1:E10.

```
Worksheets(1).Range("e1:e10").FormatConditions(1) _  
    .Modify xlCellValue, xlLess, "=$a$1"
```



[Show All](#)

# FormControlType Property

Returns the [Microsoft Excel control](#) type. Read-only [XIFormControl](#).

XIFormControl can be one of these XIFormControl constants.

**xlButtonControl**

**xlCheckBox**

**xlDropDown**

**xlEditBox**

**xlGroupBox**

**xlLabel**

**xlListBox**

**xlOptionButton**

**xlScrollBar**

**xlSpinner**

*expression*.**FormControlType**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

You cannot use this property with [ActiveX controls](#) (the **Type** property for the **Shape** object must return **msoFormControl**).

## Example

This example clears all the Microsoft Excel check boxes on worksheet one.

```
For Each s In Worksheets(1).Shapes
    If s.Type = msoFormControl Then
        If s.FormControlType = xlCheckBox Then _
            s.ControlFormat.Value = False
    End If
Next
```



[Show All](#)

# Formula Property

 [Formula property as it applies to the \*\*PivotField\*\*, \*\*PivotFormula\*\*, \*\*PivotItem\*\*, and \*\*Series\*\* objects.](#)

Returns or sets the object's formula in A1-style notation and in the language of the macro. Read/write **String**.

*expression*.**Formula**

*expression* Required. An expression that returns one of the above objects.

 [Formula property as it applies to the \*\*CalculatedMember\*\* object.](#)

Returns the member's formula in multidimensional expressions (MDX) syntax. Read-only **String**.

*expression*.**Formula**

*expression* Required. An expression that returns a **CalculatedMember** object.

 [Formula property as it applies to the \*\*Range\*\* object.](#)

Returns or sets the object's formula in A1-style notation and in the language of the macro. Read/write **Variant**.

*expression*.**Formula**

*expression* Required. An expression that returns a **Range** object.

## Remarks

This property is not available for [OLAP](#) data sources.

If the cell contains a constant, this property returns the constant. If the cell is empty, this **Formula** property returns an empty string. If the cell contains a formula, the **Formula** property returns the formula as a string in the same format that would be displayed in the formula bar (including the equal sign).

If you set the value or formula of a cell to a date, Microsoft Excel checks to see whether that cell is already formatted with one of the date or time number formats. If not, Microsoft Excel changes the number format to the default short date number format.

If the range is a one- or two-dimensional range, you can set the formula to a Visual Basic array of the same dimensions. Similarly, you can put the formula into a Visual Basic array.

Setting the formula for a multiple-cell range fills all cells in the range with the formula.

## Example

 [As it applies to the \*\*Range\*\* object.](#)

This example sets the formula for cell A1 on Sheet1.

```
Worksheets("Sheet1").Range("A1").Formula = "=$A$4+$A$10"
```



# Formula1 Property

Returns the value or expression associated with the conditional format or data validation. Can be a constant value, a string value, a cell reference, or a formula.  
Read-only **String**.

## Example

This example changes the formula for conditional format one for cells E1:E10 if the formula specifies “less than 5.”

```
With Worksheets(1).Range("e1:e10").FormatConditions(1)
    If .Operator = xlLess And .Formula1 = "5" Then
        .Modify xlCellValue, xlLess, "10"
    End If
End With
```



# Formula2 Property

Returns the value or expression associated with the second part of a conditional format or data validation. Used only when the data validation conditional format **Operator** property is **xlBetween** or **xlNotBetween**. Can be a constant value, a string value, a cell reference, or a formula. Read-only **String**.

## Example

This example changes the formula for conditional format one for cells E1:E10 if the formula specifies "between 5 and 10"

```
With Worksheets(1).Range("e1:e10").FormatConditions(1)
    If .Operator = xlBetween And _
        .Formula1 = "5" And _
        .Formula2 = "10" Then
        .Modify xlCellValue, xlLess, "10"
    End If
End With
```



# FormulaArray Property

Returns or sets the array formula of a range. Returns (or can be set to) a single formula or a Visual Basic array. If the specified range doesn't contain an array formula, this property returns **Null**. Read/write **Variant**.

## Remarks

If you use this property to enter an array formula, the formula must use the R1C1 reference style, not the A1 reference style (see the second example).

## Example

This example enters the number 3 as an array constant in cells A1:C5 on Sheet1.

```
Worksheets("Sheet1").Range("A1:C5").FormulaArray = "=3"
```

This example enters the array formula =SUM(R1C1:R3C3) in cells E1:E3 on Sheet1.

```
Worksheets("Sheet1").Range("E1:E3").FormulaArray = _  
    "=Sum(R1C1:R3C3)"
```



[Show All](#)

# FormulaHidden Property

 [FormulaHidden property as it applies to the \*\*Style\*\* object.](#)

**True** if the formula will be hidden when the worksheet is protected. Read/write **Boolean**.

*expression*.**FormulaHidden**

*expression* Required. An expression that returns a **Style** object.

 [FormulaHidden property as it applies to the \*\*CellFormat\*\* and \*\*Range\*\* objects.](#)

**True** if the formula will be hidden when the worksheet is protected. Returns **Null** if the specified range contains some cells with **FormulaHidden** equal to **True** and some cells with **FormulaHidden** equal to **False**. Read/write **Variant**.

*expression*.**FormulaHidden**

*expression* Required. An expression that returns one of the above objects.

## Remarks

Don't confuse this property with the [Hidden](#) property. The formula will not be hidden if the workbook is protected and the worksheet is not, but only if the worksheet is protected.

## Example

 [As it applies to the \*\*CellFormat\*\* and \*\*Range\*\* objects.](#)

This example hides the formulas in cells A1 and B1 on Sheet1 when the worksheet is protected.

```
Sub HideFormulas()
```

```
    Worksheets("Sheet1").Range("A1:B1").FormulaHidden = True
```

```
End Sub
```



[Show All](#)

# FormulaLabel Property

Returns or sets the formula label type for the specified range. Can be **xlNone** if the range contains no labels, or one of the following **XlFormulaLabel** constants. Read/write [XlFormulaLabel](#).

XlFormulaLabel can be one of these XlFormulaLabel constants.

**xlColumnLabels**

**xlMixedLabels**

**xlNoLabels**

**xlRowLabels**

*expression*.**FormulaLabel**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example topic sets the **AcceptLabelsInFormulas** property and then sets cells B1:D1 to be column labels.

```
ActiveWorkbook.AcceptLabelsInFormulas = True  
Worksheets(1).Range("b1:d1").FormulaLabel = xlColumnLabels
```



# FormulaLocal Property

Returns or sets the formula for the object, using A1-style references in the language of the user. Read/write **Variant** for **Range** objects, read/write **String** for **Series** objects.

## Remarks

If the cell contains a constant, this property returns that constant. If the cell is empty, the property returns an empty string. If the cell contains a formula, the property returns the formula as a string, in the same format in which it would be displayed in the formula bar (including the equal sign).

If you set the value or formula of a cell to a date, Microsoft Excel checks to see whether that cell is already formatted with one of the date or time number formats. If not, the number format is changed to the default short date number format.

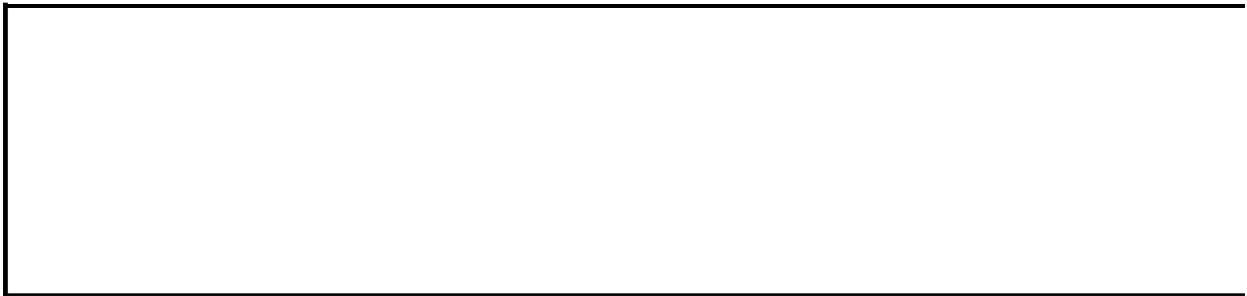
If the range is a one- or two-dimensional range, you can set the formula to a Visual Basic array of the same dimensions. Similarly, you can put the formula into a Visual Basic array.

Setting the formula of a multiple-cell range fills all cells in the range with the formula.

## Example

Assume that you enter the formula =SUM(A1:A10) in cell A11 on worksheet one, using the American English version of Microsoft Excel. If you then open the workbook on a computer that's running the German version and run the following example, the example displays the formula =SUMME(A1:A10) in a message box.

```
MsgBox Worksheets(1).Range(A11).FormulaLocal
```



# FormulaR1C1 Property

Returns or sets the formula for the object, using R1C1-style notation in the language of the macro. Read/write **Variant** for **Range** objects, read/write **String** for **Series** objects.

## Remarks

If the cell contains a constant, this property returns the constant. If the cell is empty, the property returns an empty string. If the cell contains a formula, the property returns the formula as a string, in the same format in which it would be displayed in the formula bar (including the equal sign).

If you set the value or formula of a cell to a date, Microsoft Excel checks to see whether that cell is already formatted with one of the date or time number formats. If not, the number format is changed to the default short date number format.

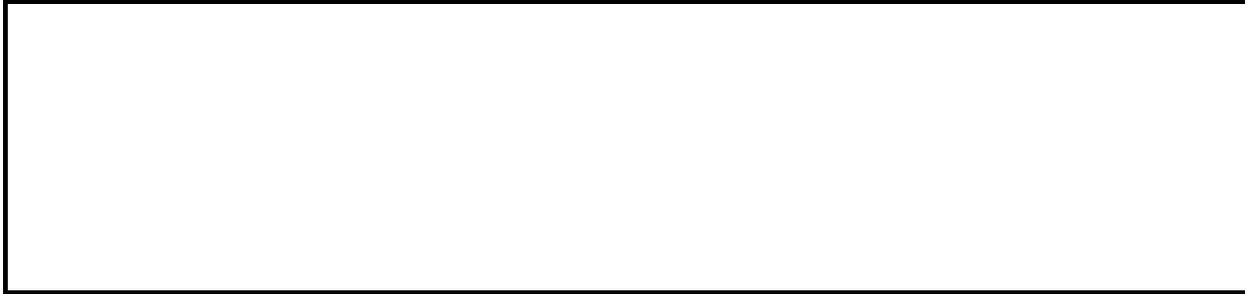
If the range is a one- or two-dimensional range, you can set the formula to a Visual Basic array of the same dimensions. Similarly, you can put the formula into a Visual Basic array.

Setting the formula of a multiple-cell range fills all cells in the range with the formula.

## Example

This example sets the formula for cell B1 on Sheet1.

```
Worksheets("Sheet1").Range("B1").FormulaR1C1 = "=SQRT(R1C1)"
```



# FormulaR1C1Local Property

Returns or sets the formula for the object, using R1C1-style notation in the language of the user. Read/write **Variant** for **Range** objects, read/write **String** for **Series** objects.

## Remarks

If the cell contains a constant, this property returns that constant. If the cell is empty, the property returns an empty string. If the cell contains a formula, the property returns the formula as a string, in the same format in which it would be displayed in the formula bar (including the equal sign).

If you set the value or formula of a cell to a date, Microsoft Excel checks to see whether that cell is already formatted with one of the date or time number formats. If not, the number format is changed to the default short date number format.

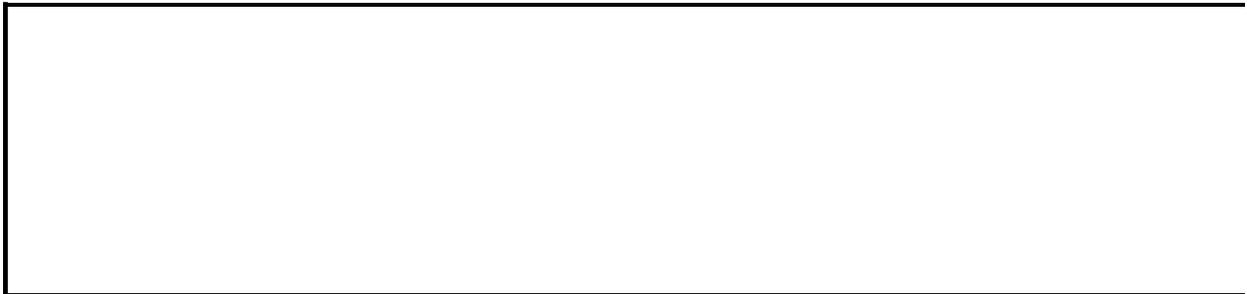
If the range is a one- or two-dimensional range, you can set the formula to a Visual Basic array of the same dimensions. Similarly, you can put the formula into a Visual Basic array.

Setting the formula of a multiple-cell range fills all cells in the range with the formula.

## Example

Assume that you enter the formula =SUM(A1:A10) in cell A11 on worksheet one, using the American English version of Microsoft Excel. If you then open the workbook on a computer that's running the German version and run the following example, the example displays the formula =SUMME(Z1S1:Z10S1) in a message box.

```
MsgBox Worksheets(1).Range("A11").FormulaR1C1Local
```



# Forward Property

Returns or sets the number of periods (or units on a scatter chart) that the trendline extends forward. Read/write **Long**.

## Example

This example sets the number of units that the trendline on Chart1 extends forward and backward. The example should be run on a 2-D column chart that contains a single series with a trendline.

```
With Charts("Chart1").SeriesCollection(1).Trendlines(1)  
    .Forward = 5  
    .Backward = .5  
End With
```



# FreezePanels Property

**True** if split panes are frozen. Read/write **Boolean**.

## Remarks

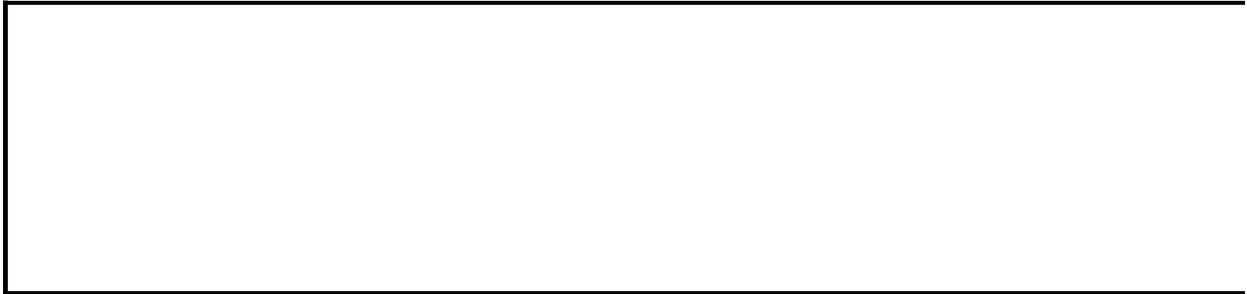
It's possible for **FreezePanels** to be **True** and **Split** to be **False**, or vice versa.

This property applies only to worksheets and macro sheets.

## Example

This example freezes split panes in the active window in Book1.xls.

```
Workbooks("BOOK1.XLS").Worksheets("Sheet1").Activate  
ActiveWindow.FreezePanes = True
```



# FullName Property

Returns the name of the object, including its path on disk, as a string. Read-only **String**.

## Remarks

This property is equivalent to the [Path](#) property, followed by the current file system separator, followed by the [Name](#) property.

## Example

This example displays the path and file name of every available add-in.

```
For Each a In AddIns  
    MsgBox a.FullName  
Next a
```

This example displays the path and file name of the active workbook (assuming that the workbook has been saved).

```
MsgBox ActiveWorkbook.FullName
```



# FullNameURLEncoded Property

Returns a **String** indicating the name of the object, including its path on disk, as a string. Read-only.

*expression*.**FullNameURLEncoded**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel displays the path and file name of the active workbook to the user.

```
Sub UseCanonical()  
    ' Display the full path to user.  
    MsgBox ActiveWorkbook.FullNameURLEncoded  
End Sub
```



[Show All](#)

# Function Property

Returns or sets the function used to summarize the PivotTable field (data fields only). Read/write [XLConsolidationFunction](#).

XLConsolidationFunction can be one of these XLConsolidationFunction constants.

**xlAverage**

**xlCountNums**

**xlMin**

**xlStDev**

**xlSum**

**xlVar**

**xlCount**

**xlMax**

**xlProduct**

**xlStDevP**

**xlUnknown**

**xlVarP**

*expression*.**Function**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

For [OLAP](#) data sources, this property is read-only and always returns **xlUnknown**. For other data sources, this property cannot be set to **xlUnknown**.

## Example

This example sets the Sum of 1994 field in the first PivotTable report on the active sheet to use the SUM function.

```
ActiveSheet.PivotTables("PivotTable1") _  
    .PivotFields("Sum of 1994").Function = xlSum
```



# Gap Property

Returns or sets the horizontal distance (in points) between the end of the callout line and the text bounding box. Read/write **Single**.

## Example

This example sets the distance between the callout line and the text bounding box to 3 points for shape one on myDocument. For the example to work, shape one must be a callout.

```
Set myDocument = Worksheets(1)  
myDocument.Shapes(1).Callout.Gap = 3
```



# GapDepth Property

Returns or sets the distance between the data series in a 3-D chart, as a percentage of the marker width. The value of this property must be between 0 and 500. Read/write **Long**.

## Example

This example sets the distance between the data series in Chart1 to 200 percent of the marker width. The example should be run on a 3-D chart (the **GapDepth** property fails on 2-D charts).

```
Charts("Chart1").GapDepth = 200
```



# GapWidth Property

Bar and Column charts: Returns or sets the space between bar or column clusters, as a percentage of the bar or column width. The value of this property must be between 0 and 500. Read/write **Long**.

Pie of Pie and Bar of Pie charts: Returns or sets the space between the primary and secondary sections of the chart. The value of this property must be between 5 and 200. Read/write **Long**.

## Example

This example sets the space between column clusters in Chart1 to be 50 percent of the column width.

```
Charts("Chart1").ChartGroups(1).GapWidth = 50
```



# GenerateGetPivotData Property

Returns **True** when Microsoft Excel can get PivotTable report data. Read/write **Boolean**.

*expression*.**GenerateGetPivotData**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In the following example, Microsoft Excel determines the status of getting PivotTable report data and notifies the user. This example assumes a PivotTable report exists on the active worksheet.

```
Sub PivotTableInfo()  
    ' Determine the ability to get PivotTable report data and notify  
    If Application.GenerateGetPivotData = True Then  
        MsgBox "The ability to get PivotTable report data is enabled"  
    Else  
        MsgBox "The ability to get PivotTable report data is disable"  
    End If  
End Sub
```



# GermanPostReform Property

**True** to check the spelling of words using the German post-reform rules. **False** cancels this feature. Read/write **Boolean**.

*expression*.**GermanPostReform**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel determines if the checking of spelling for German words is using post-reform rules and enables this feature if it's not enabled, and then notifies the user of the status.

```
Sub SpellingCheck()  
    ' Determine if spelling check for German words is using post-ref  
    If Application.SpellingOptions.GermanPostReform = False Then  
        Application.SpellingOptions.GermanPostReform = True  
        MsgBox "German words will now use post-reform rules."  
    Else  
        MsgBox "German words using post-reform rules has already bee  
    End If  
End Sub
```



[Show All](#)

# GradientColorType Property

Returns the gradient color type for the specified fill. Read-only [MsoGradientColorType](#).

MsoGradientColorType can be one of these MsoGradientColorType constants.

**msoGradientColorMixed**

**msoGradientOneColor**

**msoGradientPresetColors**

**msoGradientTwoColors**

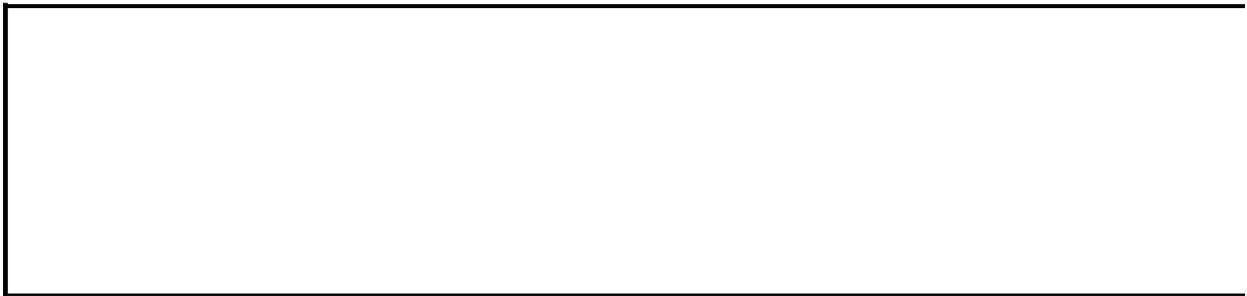
*expression*.**GradientColorType**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example sets the fill format for chart two to the same style used for chart one.

```
Set c1f = Charts(1).ChartArea.Fill
If c1f.Type = msoFillGradient And _
    c1f.GradientColorType = msoGradientOneColor Then
    With Charts(2).ChartArea.Fill
        .Visible = True
        .OneColorGradient c1f.GradientStyle, _
            c1f.GradientVariant, c1f.GradientDegree
    End With
End If
```



# GradientDegree Property

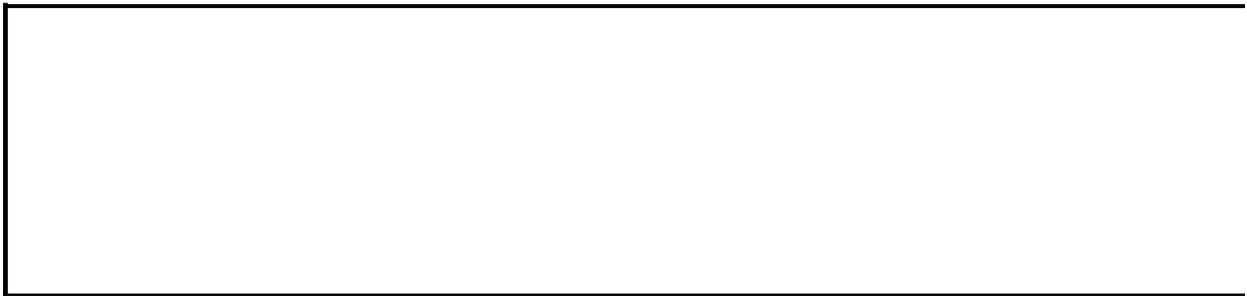
Returns the gradient degree of the specified one-color shaded fill as a floating-point value from 0.0 (dark) through 1.0 (light). Read-only **Single**.

This property is read-only. Use the **OneColorGradient** method to set the gradient degree for the fill.

## Example

This example sets the fill format for chart two to the same style used for chart one.

```
Set c1f = Charts(1).ChartArea.Fill
If c1f.Type = msoFillGradient And _
    c1f.GradientColorType = msoGradientOneColor Then
    With Charts(2).ChartArea.Fill
        .Visible = True
        .OneColorGradient c1f.GradientStyle, _
            c1f.GradientVariant, c1f.GradientDegree
    End With
End If
```



# GradientVariant Property

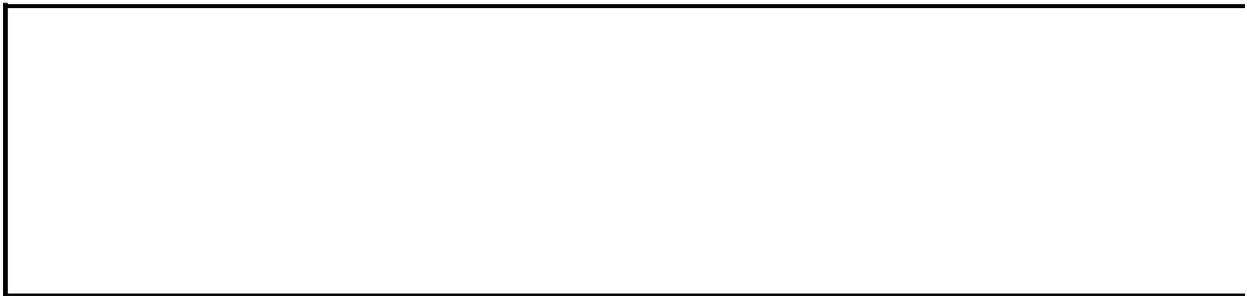
Returns the shade variant for the specified fill as an integer value from 1 through 4. The values for this property correspond to the gradient variants (numbered from left to right and from top to bottom) on the **Gradient** tab in the **Fill Effects** dialog box. Read-only **Long**.

This property is read-only. Use the **OneColorGradient** or **TwoColorGradient** method to set the gradient variant for the fill.

## Example

This example sets the fill format for chart two to the same style used for chart one.

```
Set c1f = Charts(1).ChartArea.Fill
If c1f.Type = msoFillGradient And _
    c1f.GradientColorType = msoGradientOneColor Then
    With Charts(2).ChartArea.Fill
        .Visible = True
        .OneColorGradient c1f.GradientStyle, _
            c1f.GradientVariant, c1f.GradientDegree
    End With
End If
```



# GrandTotalName Property

Returns or sets the text string label that is displayed in the grand total column or row heading in the specified PivotTable report. The default value is the string "Grand Total". Read/write **String**.

## Example

This example sets the grand total heading label to "Regional Total" in the second PivotTable report on the active worksheet.

```
ActiveSheet.PivotTables("PivotTable2").GrandTotalName = "Regional To
```



# GridlineColor Property

Returns or sets the gridline color as an RGB value. Read/write **Long**.

## Example

This example sets the gridline color in the active window in Book1.xls to red.

```
Workbooks("BOOK1.XLS").Worksheets("Sheet1").Activate  
ActiveWindow.GridlineColor = RGB(255,0,0)
```



[Show All](#)

# GridlineColorIndex Property

Returns or sets the gridline color as an index into the current color palette or as the following [XIColorIndex](#) constant.

XIColorIndex can be the following XIColorIndex constant.

**xIColorIndexAutomatic**

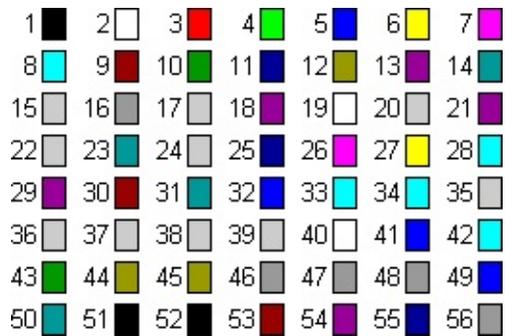
*expression*.**GridlineColorIndex**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Set this property to **xlColorIndexAutomatic** to specify the automatic color.

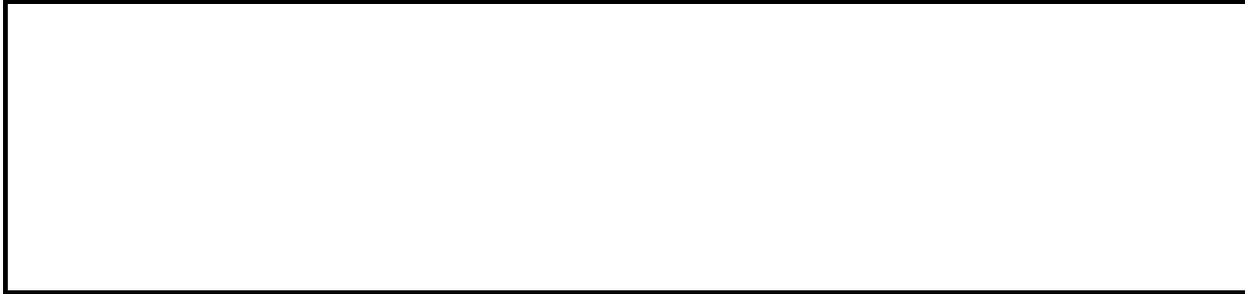
The following illustration shows the color-index values in the default color palette.



## Example

This example sets the gridline color in the active window to blue.

```
ActiveWindow.GridlineColorIndex = 5
```



# GroupItems Property

Returns a [GroupShapes](#) object that represents the individual shapes in the specified group. Use the [Item](#) method of the **GroupShapes** object to return a single shape from the group. Applies to **Shape** or **ShapeRange** objects that represent grouped shapes. Read-only.

## Example

This example adds three triangles to myDocument, groups them, sets a color for the entire group, and then changes the color for the second triangle only.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes
    .AddShape(msoShapeIsoscelesTriangle, _
        10, 10, 100, 100).Name = "shpOne"
    .AddShape(msoShapeIsoscelesTriangle, _
        150, 10, 100, 100).Name = "shpTwo"
    .AddShape(msoShapeIsoscelesTriangle, _
        300, 10, 100, 100).Name = "shpThree"
    With .Range(Array("shpOne", "shpTwo", "shpThree")).Group
        .Fill.PresetTextured msoTextureBlueTissuePaper
        .GroupItems(2).Fill.PresetTextured msoTextureGreenMarble
    End With
End With
```



[Show All](#)

# GroupLevel Property

Returns the placement of the specified field within a group of fields (if the field is a member of a grouped set of fields). Read-only.

## Remarks

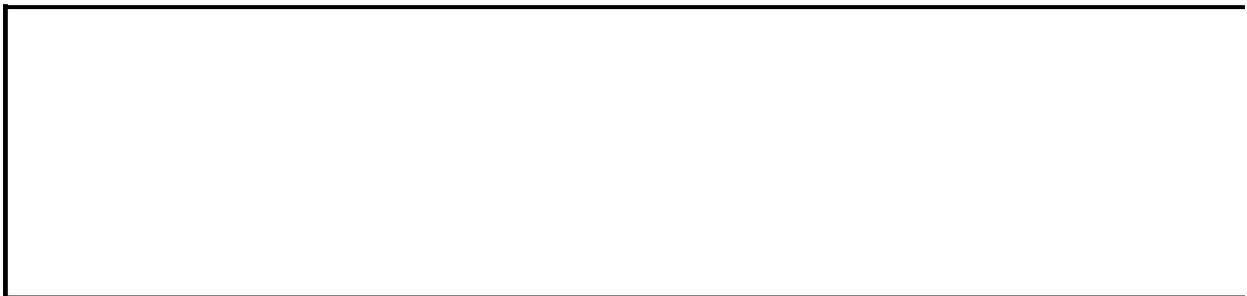
This property is not available for [OLAP](#) data sources.

The highest-level parent field (leftmost parent field) is level one, its child is level two, and so on.

## Example

This example displays a message box if the field that contains the active cell is the highest-level parent field.

```
Worksheets("Sheet1").Activate  
If ActiveCell.PivotField.GroupLevel = 1 Then  
    MsgBox "This is the highest-level parent field."  
End If
```



# Has3DEffect Property

**True** if the series has a three-dimensional appearance. Applies only to bubble charts. Read/write **Boolean**.

## Example

This example gives series one on the embedded bubble chart a three-dimensional appearance.

```
With Worksheets(1).ChartObjects(1).Chart  
    .SeriesCollection(1).Has3DEffect = True  
End With
```



# Has3DShading Property

**True** if the chart group has three-dimensional shading. This property only applies to surface charts and will return a run-time error if you try to set it to a non-surface chart. Read/write **Boolean**.

## Example

This example adds three-dimensional shading to a chart group.

```
Charts(1).ChartGroups(1).Has3DShading = True
```



# HasArray Property

**True** if the specified cell is part of an array formula. Read-only **Variant**.

## Example

This example displays a message if the active cell on Sheet1 is part of an array.

```
Worksheets("Sheet1").Activate  
If ActiveCell.HasArray =True Then  
    MsgBox "The active cell is part of an array"  
End If
```



# HasAutoFormat Property

**True** if the PivotTable report is automatically formatted when it's refreshed or when fields are moved. Read/write **Boolean**.

## Example

This example causes the PivotTable report to be automatically reformatted when it's refreshed or when fields are moved.

```
Set pvtTable = Worksheets("Sheet1").Range("A3").PivotTable  
pvtTable.HasAutoFormat = True
```



[Show All](#)

# HasAxis Property

Returns or sets which axes exist on the chart. Read/write **Variant**.

*expression*.**HasAxis**(*Index1*, *Index2*)

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Index1** Optional **Variant**. The axis type. Series axes apply only to 3-D charts. Can be one of the [XlAxisType](#) constants.

XlAxisType can be one of the following XlAxisType constants.

**xlCategory**

**xlValue**

**xlSeriesAxis**

**Index2** Optional **Variant**. The axis group. 3-D charts have only one set of axes. Can be one of the [XlAxisGroup](#) constants.

XlAxisGroup can be one of the following XlAxisGroup constants.

**xlPrimary**

**xlSecondary**

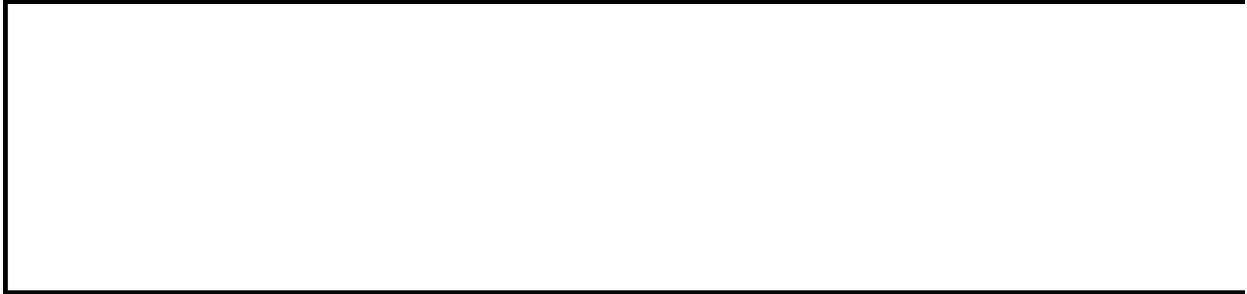
## Remarks

Microsoft Excel may create or delete axes if you change the chart type or the [AxisGroup](#) property.

## Example

This example turns on the primary value axis for Chart1.

```
Charts("Chart1").HasAxis(xlValue, xlPrimary) = True
```



# HasBorderHorizontal Property

**True** if the chart data table has horizontal cell borders. Read/write **Boolean**.

## Example

This example causes the embedded chart data table to be displayed with an outline border and no cell borders.

```
With Worksheets(1).ChartObjects(1).Chart
    .HasDataTable = True
    With .DataTable
        .HasBorderHorizontal = False
        .HasBorderVertical = False
        .HasBorderOutline = True
    End With
End With
```



# HasBorderOutline Property

**True** if the chart data table has outline borders. Read/write **Boolean**.

## Example

This example causes the embedded chart data table to be displayed with an outline border and no cell borders.

```
With Worksheets(1).ChartObjects(1).Chart
    .HasDataTable = True
    With .DataTable
        .HasBorderHorizontal = False
        .HasBorderVertical = False
        .HasBorderOutline = True
    End With
End With
```



# HasBorderVertical Property

**True** if the chart data table has vertical cell borders. Read/write **Boolean**.

## Example

This example causes the embedded chart data table to be displayed with an outline border and no cell borders.

```
With Worksheets(1).ChartObjects(1).Chart
    .HasDataTable = True
    With .DataTable
        .HasBorderHorizontal = False
        .HasBorderVertical = False
        .HasBorderOutline = True
    End With
End With
```



# HasDataLabel Property

**True** if the point has a data label. Read/write **Boolean**.

## Example

This example turns on the data label for point seven in series three in Chart1, and then it sets the data label color to blue.

```
With Charts("Chart1").SeriesCollection(3).Points(7)  
    .HasDataLabel = True  
    .ApplyDataLabels Type:=xlValue  
    .DataLabel.Font.ColorIndex = 5  
End With
```



# HasDataLabels Property

**True** if the series has data labels. Read/write **Boolean**.

## Example

This example turns on data labels for series three in Chart1.

```
With Charts("Chart1").SeriesCollection(3)  
    .HasDataLabels = True  
    .ApplyDataLabels Type:=xlValue  
End With
```



# HasDataTable Property

**True** if the chart has a data table. Read/write **Boolean**.

## Example

This example causes the embedded chart data table to be displayed with an outline border and no cell borders.

```
With Worksheets(1).ChartObjects(1).Chart
    .HasDataTable = True
    With .DataTable
        .HasBorderHorizontal = False
        .HasBorderVertical = False
        .HasBorderOutline = True
    End With
End With
```



[Show All](#)

# HasDiagram Property

Returns whether a shape or shape range contains a diagram. Read-only [MsoTriState](#).

MsoTriState can be one of these MsoTriState constants.

**msoCTrue** Not used for this property.

**msoFalse** Returned if a shape is not a diagram.

**msoTriStateMixed** Not used for this property.

**msoTriStateToggle** Not used for this property.

**msoTrue** Returned if a shape is a diagram.

*expression*.**HasDiagram**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

The following example places a diagram in the active worksheet and then displays a message to tell the user whether the diagram was successfully created or not.

```
Sub CheckforDiagram()  
  
    Dim shDiagram As Shape  
  
    Set shDiagram = ActiveSheet.Shapes.AddDiagram( _  
        Type:=msoDiagramOrgChart, Top:=10, Left:=15, _  
        Width:=400, Height:=475)  
  
    ' Notify user about diagram.  
    If shDiagram.HasDiagram = msoTrue Then  
        MsgBox "A diagram is present."  
    Else  
        MsgBox "No diagram is present."  
    End If  
  
End Sub
```



[Show All](#)

# HasDiagramNode Property

Returns a value indicating whether a diagram node exists in a given shape or shape range. Read-only [MsoTriState](#).

MsoTriState can be one of these MsoTriState constants.

**msoCTrue** Not used for this property.

**msoFalse** Returns if a shape is not a diagram node.

**msoTriStateMixed** Not used for this property.

**msoTriStateToggle** Not used for this property.

**msoTrue** Returns if a shape is a diagram node.

*expression*.**HasDiagramNode**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

The following example places a diagram node in the active worksheet and then displays a message to notify the user whether or not the node was successfully created.

```
Sub IsDiagram()  
  
    Dim shDiagram As Shape  
    Dim nodItem As DiagramNode  
  
    Set shDiagram = ActiveSheet.Shapes.AddDiagram( _  
        Type:=msoDiagramOrgChart, Top:=10, _  
        Left:=15, Width:=400, Height:=475)  
    Set nodItem = shDiagram.DiagramNode  
  
    'Add a root node to the diagram.  
    nodItem.Children.AddNode  
  
    ' Notify user about diagram.  
    If shDiagram.HasDiagramNode = msoTrue Then  
        MsgBox "Diagram node present"  
    Else  
        MsgBox "No diagram node present"  
    End If  
  
End Sub
```



# HasDisplayUnitLabel Property

**True** if the label specified by the [DisplayUnit](#) or [DisplayUnitCustom](#) property is displayed on the specified axis. The default value is **True**. Read/write **Boolean**.

## Example

This example sets the units on the value axis in Chart1 to increments of 500 but keeps the unit label hidden.

```
With Charts("Chart1").Axes(xlValue)  
    .DisplayUnit = xlCustom  
    .DisplayUnitCustom = 500  
    .AxisTitle.Caption = "Rebate Amounts"  
    .HasDisplayUnitLabel = False  
End With
```



# HasDropLines Property

**True** if the line chart or area chart has drop lines. Applies only to line and area charts. Read/write **Boolean**.

## Example

This example turns on drop lines for chart group one in Chart1 and then sets their line style, weight, and color. The example should be run on a 2-D line chart that has one series.

```
With Charts("Chart1").ChartGroups(1)
    .HasDropLines = True
    With .DropLines.Border
        .LineStyle = xlThin
        .Weight = xlMedium
        .ColorIndex = 3
    End With
End With
```



# HasErrorBars Property

**True** if the series has error bars. This property isn't available for 3-D charts.  
Read/write **Boolean**.

## Example

This example removes error bars from series one in Chart1. The example should be run on a 2-D line chart that has error bars for series one.

```
Charts("Chart1").SeriesCollection(1).HasErrorBars = False
```



# HasFormula Property

**True** if all cells in the range contain formulas; **False** if none of the cells in the range contains a formula; **Null** otherwise. Read-only **Variant**.

## Example

This example prompts the user to select a range on Sheet1. If every cell in the selected range contains a formula, the example displays a message.

```
Worksheets("Sheet1").Activate
Set rr = Application.InputBox( _
    prompt:="Select a range on this worksheet", _
    Type:=8)
If rr.HasFormula = True Then
    MsgBox "Every cell in the selection contains a formula"
End If
```



# HasHiLoLines Property

**True** if the line chart has high-low lines. Applies only to line charts. Read/write **Boolean**.

## Example

This example turns on high-low lines for chart group one in Chart1 and then sets line style, weight, and color. The example should be run on a 2-D line chart that has three series of stock-quote-like data (high-low-close).

```
With Charts("Chart1").ChartGroups(1)
    .HasHiLoLines = True
    With .HiLoLines.Border
        .LineStyle = xlThin
        .Weight = xlMedium
        .ColorIndex = 3
    End With
End With
```



# HasLeaderLines Property

**True** if the series has leader lines. Read/write **Boolean**.

## Example

This example adds data labels and blue leader lines to series one on the pie chart.

```
With Worksheets(1).ChartObjects(1).Chart.SeriesCollection(1)
    .HasDataLabels = True
    .DataLabels.Position = xlLabelPositionBestFit
    .HasLeaderLines = True
    .LeaderLines.Border.ColorIndex = 5
End With
```



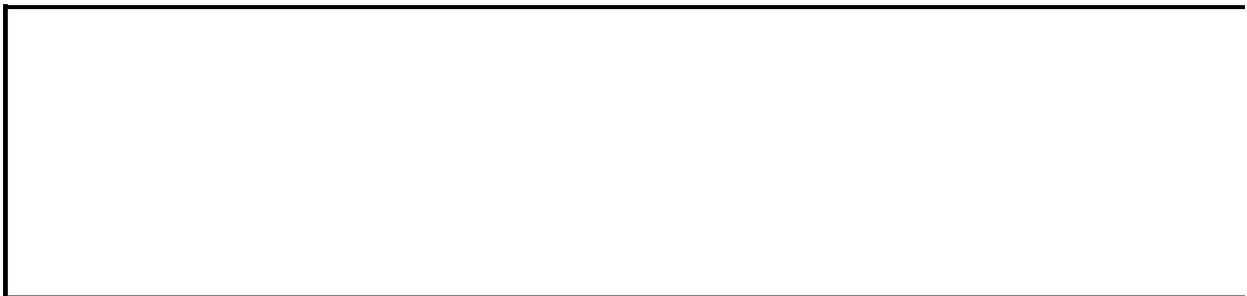
# HasLegend Property

**True** if the chart has a legend. Read/write **Boolean**.

## Example

This example turns on the legend for Chart1 and then sets the legend font color to blue.

```
With Charts("Chart1")  
    .HasLegend = True  
    .Legend.Font.ColorIndex = 5  
End With
```



# HasMajorGridlines Property

**True** if the axis has major gridlines. Only axes in the primary axis group can have gridlines. Read/write **Boolean**.

## Example

This example sets the color of the major gridlines for the value axis in Chart1.

```
With Charts("Chart1").Axes(xlValue)
    If .HasMajorGridlines Then
        .MajorGridlines.Border.ColorIndex = 3    'set color to red
    End If
End With
```



# HasMemberProperties Property

Returns **True** when there are member properties specified to be displayed for the cube field. Read-only **Boolean**.

*expression*.**HasMemberProperties**

*expression* Required. An expression that returns a [CubeField](#) object.

## Example

The example determines if there are member properties to be displayed for the cube field and notifies the user. The example assumes a PivotTable exists on the active worksheet.

```
Sub UseHasMemberProperties()  
  
    Dim pvtTable As PivotTable  
    Dim cbeField As CubeField  
  
    Set pvtTable = ActiveSheet.PivotTables(1)  
    Set cbeField = pvtTable.CubeFields("[Country]")  
  
    ' Determine if there are member properties to be displayed.  
    If cbeField.HasMemberProperties = True Then  
        MsgBox "There are member properties to be displayed."  
    Else  
        MsgBox "There are no member properties to be displayed."  
    End If  
  
End Sub
```



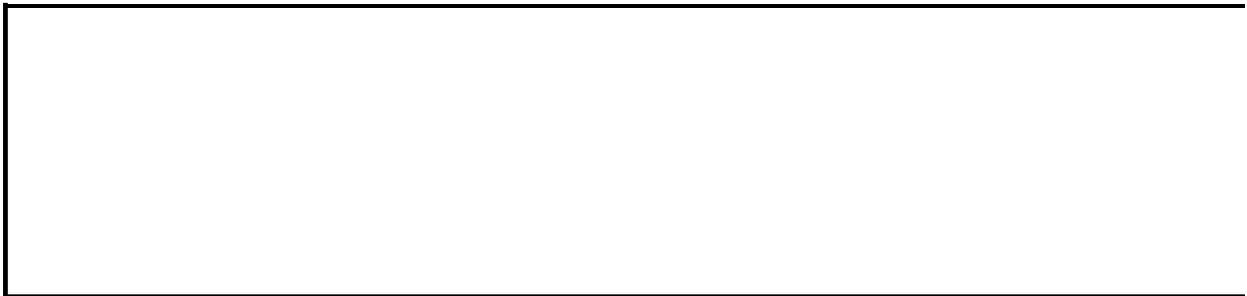
# HasMinorGridlines Property

**True** if the axis has minor gridlines. Only axes in the primary axis group can have gridlines. Read/write **Boolean**.

## Example

This example sets the color of the minor gridlines for the value axis in Chart1.

```
With Charts("Chart1").Axes(xlValue)
    If .HasMinorGridlines Then
        .MinorGridlines.Border.ColorIndex = 4
        'set color to green
    End If
End With
```



# HasPassword Property

**True** if the workbook has a protection password. Read-only **Boolean**.

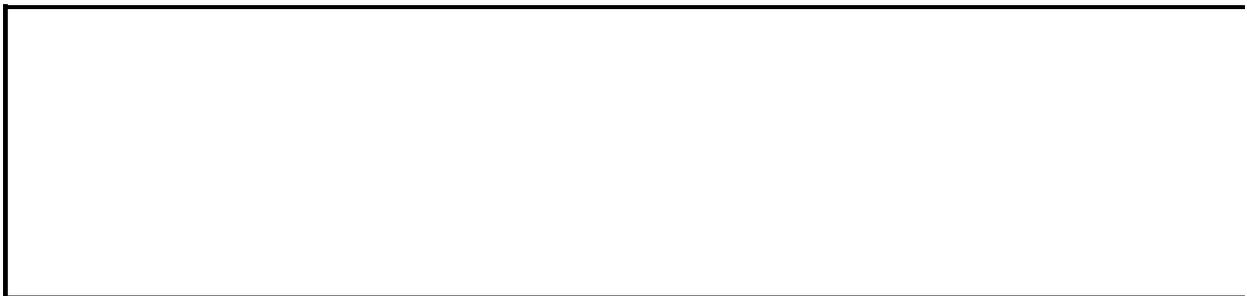
## Remarks

You can assign a protection password to a workbook by using the [SaveAs](#) method.

## Example

This example displays a message if the active workbook has a protection password.

```
If ActiveWorkbook.HasPassword = True Then  
    MsgBox "Remember to obtain the workbook password" & Chr(13) & _  
        " from the Network Administrator."  
End If
```



# HasPivotFields Property

**True** if the PivotChart controls are displayed on the specified PivotChart report. The default value is **True**. For a regular chart, this property always returns **False** and cannot be set. Read/write **Boolean**.

## Example

This example disables the PivotChart controls on the Sales chart in the 1996 Report workbook.

```
Workbooks("1996 Report").Charts("Sales") _  
    .PivotLayout.HasPivotFields = False
```



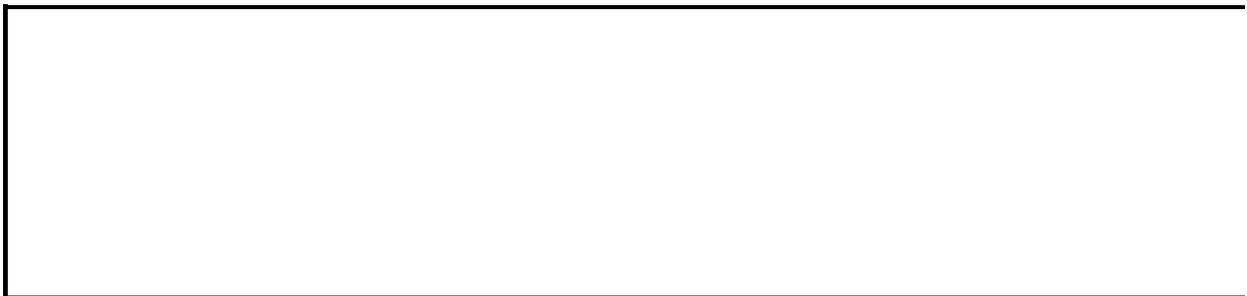
# HasRadarAxisLabels Property

**True** if a radar chart has axis labels. Applies only to radar charts. Read/write **Boolean**.

## Example

This example turns on radar axis labels for chart group one in Chart1 and sets their color. The example should be run on a radar chart.

```
With Charts("Chart1").ChartGroups(1)  
    .HasRadarAxisLabels = True  
    .RadarAxisLabels.Font.ColorIndex = 3  
End With
```



# HasRoutingSlip Property

**True** if the workbook has a routing slip. Read/write **Boolean**.

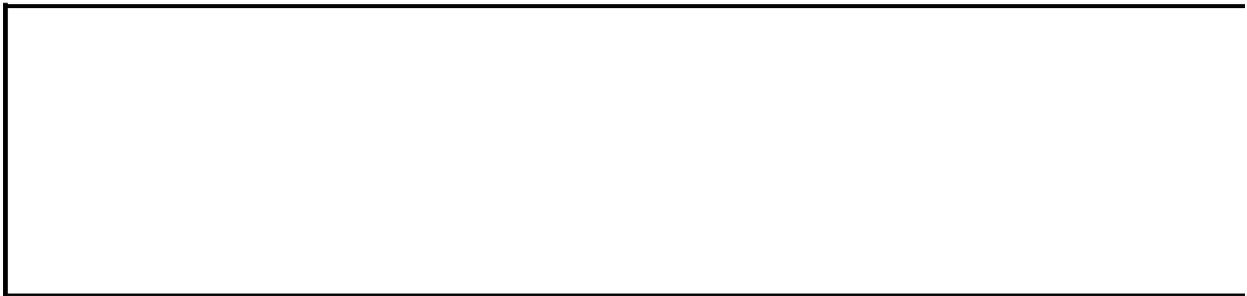
## Remarks

Setting this property to **True** creates a routing slip with default values. Setting the property to **False** deletes the routing slip.

## Example

This example creates a routing slip for Book1.xls and then sends the workbook to three recipients, one after another.

```
Workbooks("BOOK1.XLS").HasRoutingSlip = True
With Workbooks("BOOK1.XLS").RoutingSlip
    .Delivery = xlOneAfterAnother
    .Recipients = Array("Adam Bendel", _
        "Jean Selva", "Bernard Gabor")
    .Subject = "Here is BOOK1.XLS"
    .Message = "Here is the workbook. What do you think?"
End With
Workbooks("BOOK1.XLS").Route
```



# HasSeriesLines Property

**True** if a stacked column chart or bar chart has series lines or if a Pie of Pie chart or Bar of Pie chart has connector lines between the two sections. Applies only to stacked column charts, bar charts, Pie of Pie charts, or Bar of Pie charts.

Read/write **Boolean**.

## Example

This example turns on series lines for chart group one in Chart1 and then sets their line style, weight, and color. The example should be run on a 2-D stacked column chart that has two or more series.

```
With Charts("Chart1").ChartGroups(1)
    .HasSeriesLines = True
    With .SeriesLines.Border
        .LineStyle = xlThin
        .Weight = xlMedium
        .ColorIndex = 3
    End With
End With
```



# HasTitle Property

**True** if the axis or chart has a visible title. Read/write **Boolean**.

## Remarks

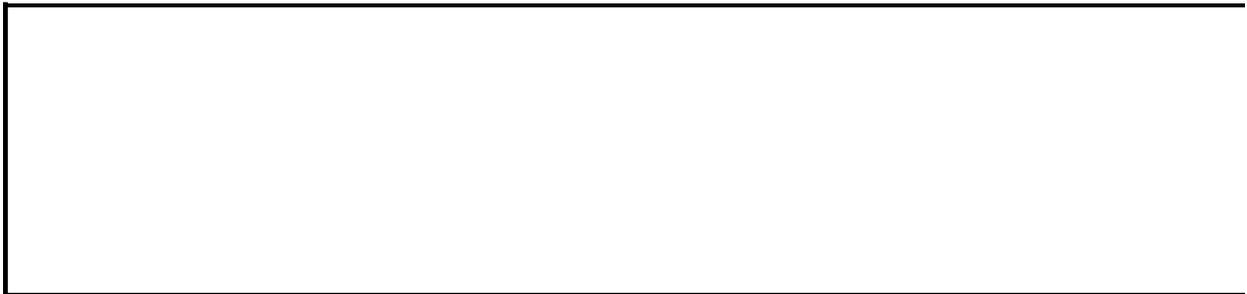
An axis title is represented by an [AxisTitle](#) object.

A chart title is represented by a [ChartTitle](#) object.

## Example

This example adds an axis label to the category axis in Chart1.

```
With Charts("Chart1").Axes(xlCategory)
    .HasTitle = True
    .AxisTitle.Text = "July Sales"
End With
```



# HasUpDownBars Property

**True** if a line chart has up and down bars. Applies only to line charts. Read/write **Boolean**.

## Example

This example turns on up and down bars for chart group one in Chart1 and then sets their colors. The example should be run on a 2-D line chart containing two series that cross each other at one or more data points.

```
With Charts("Chart1").ChartGroups(1)
    .HasUpDownBars = True
    .DownBars.Interior.ColorIndex = 3
    .UpBars.Interior.ColorIndex = 5
End With
```



[Show All](#)

# HeaderMargin Property

Returns or sets the distance from the top of the page to the header, in [points](#).  
Read/write **Double**.

## Remarks

Margins are set or returned in points. Use the **InchesToPoints** method or the **CentimetersToPoints** method to convert measurements from inches or centimeters.

## Example

This example sets the header margin of Sheet1 to 0.5 inch.

```
Worksheets("Sheet1").PageSetup.HeaderMargin = _  
    Application.InchesToPoints(0.5)
```



[Show All](#)

# HeaderRowRange Property

Returns a **Range** object that represents the range of the header row for a [list](#).  
Read-only **Range**.

*expression*.**HeaderRowRange**

*expression* Required. An expression that returns a **Range** object.

## Example

The following example activates the range specified by the **HeaderRowRange** property of the default **ListObject** object in the first worksheet of the active workbook.

```
Sub ActivateHeaderRow()  
    Dim wrksht As Worksheet  
    Dim objList As ListObject  
    Dim objListRng As Range  
  
    Set wrksht = ActiveWorkbook.Worksheets("Sheet1")  
    Set objList = wrksht.ListObjects(1)  
    Set objListRng = objList.HeaderRowRange  
  
    objListRng.Activate  
End Sub
```



# HeartbeatInterval Property

Returns or sets a **Long** for the interval between updates for real-time data.  
Read/write.

*expression*.**HeartbeatInterval**

*expression* Required. An expression that returns an **IRTDUpdateEvent** object.

## Remarks

Setting the **HeartbeatInterval** property to -1 will result in the [Heartbeat](#) method not being called.

**Note** The heartbeat interval cannot be set below 15,000 milliseconds, due to the standard 15-second time out.

---

[Show All](#)

# HebrewModes Property

Returns or sets the mode for the Hebrew spelling checker. Read/write [XlHebrewModes](#).

XlHebrewModes can be one of these XlHebrewModes constants.

**xlHebrewFullScript** (default) The conventional script type as required by the Hebrew Language Academy when writing non-diacritized text.

**xlHebrewMixedAuthorizedScript** The Hebrew traditional script.

**xlHebrewMixedScript** In this mode the speller accepts any word recognized as Hebrew, whether in Full Script, Partial Script, or any non-conventional spelling variation that is known to the speller.

**xlHebrewPartialScript** In this mode the speller accepts words both in Full Script and Partial Script. Some words will be flagged since this spelling is not authorized in either Full script or Partial script.

*expression*.**HebrewModes**

*expression* Required. An expression that returns one of the objects in the Applies To list.

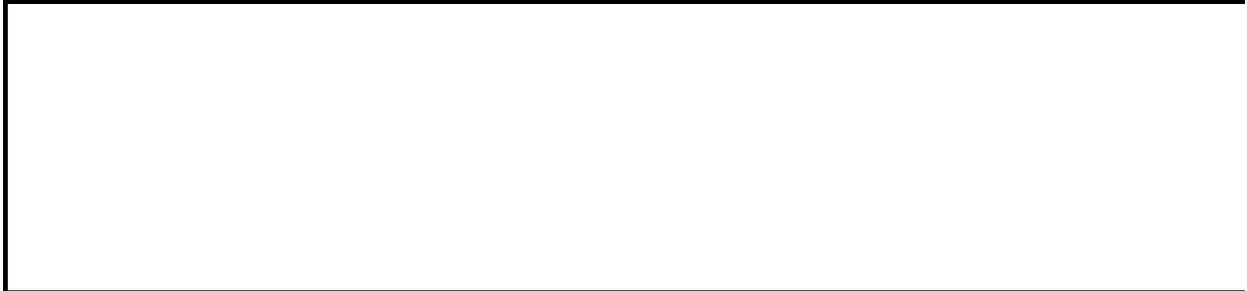
## **Remarks**

A legitimate Hebrew word can be a basic dictionary entry or any inflection.

## Example

In this example, Microsoft Excel determines the setting for the Hebrew spelling mode and notifies the user.

```
Sub CheckHebrewMode()  
    ' Determine the Hebrew spelling mode setting and notify user.  
    Select Case Application.SpellingOptions.HebrewModes  
        Case xlHebrewFullScript  
            MsgBox "The Hebrew spelling mode setting is Full Script."  
        Case xlHebrewMixedAuthorizedScript  
            MsgBox "The Hebrew spelling mode setting is Mixed Author  
        Case xlHebrewMixedScript  
            MsgBox "The Hebrew spelling mode setting is Mixed Script  
        Case xlHebrewPartialScript  
            MsgBox "The Hebrew spelling mode setting is Partial Scri  
    End Select  
End Sub
```



[Show All](#)

# Height Property

[Height property as it applies to the \*\*Application\*\* object.](#)

The height of the main application window. If the window is minimized, this property is read-only and refers to the height of the icon. If the window is maximized, this property cannot be set. Use the [WindowState](#) property to determine the window state. Read/write **Double**.

*expression*.**Height**

*expression* Required. An expression that returns an **Application** object.

[Height property as it applies to the \*\*Window\*\* object.](#)

The height of the window. Use the [UsableHeight](#) property to determine the maximum size for the window. You cannot set this property if the window is maximized or minimized. Use the [WindowState](#) property to determine the window state. Read/write **Double**.

*expression*.**Height**

*expression* Required. An expression that returns a **Window** object.

[Height property as it applies to the \*\*ChartArea\*\*, \*\*ChartObject\*\*, \*\*ChartObjects\*\*, \*\*Legend\*\*, \*\*OLEObject\*\*, \*\*OLEObjects\*\*, and \*\*PlotArea\*\* objects.](#)

The height of the object. Read/write **Double**.

*expression*.**Height**

*expression* Required. An expression that returns one of the above objects.

[Height property as it applies to the \*\*Axis\*\*, \*\*LegendEntry\*\*, and \*\*LegendKey\*\* objects.](#)

The height of the object. Read-only **Double**.

*expression*.**Height**

*expression* Required. An expression that returns one of the above objects.

[Height property as it applies to the \*\*Graphic\*\*, \*\*Shape\*\*, and \*\*ShapeRange\*\* objects.](#)

The height of the object. Read/write **Single**.

*expression*.**Height**

*expression* Required. An expression that returns one of the above objects.

[Height property as it applies to the \*\*Range\*\* object.](#)

The height of the range. Read-only **Variant**.

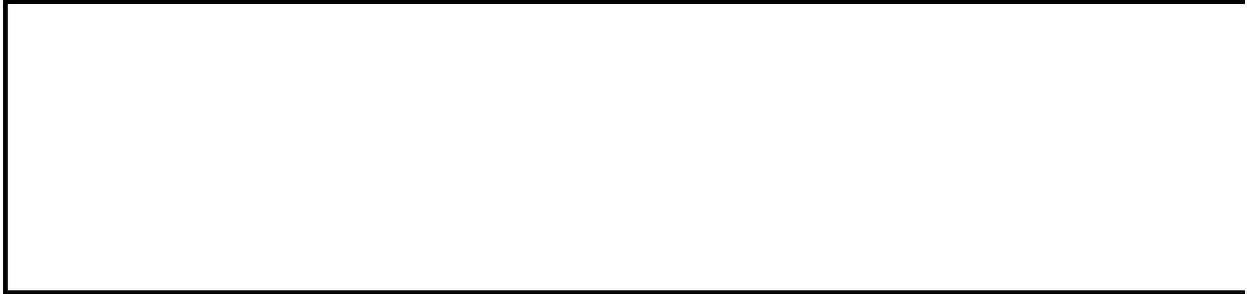
*expression*.**Height**

*expression* Required. An expression that returns a **Range** object.

## Example

This example sets the height of the embedded chart.

```
worksheets("Sheet1").ChartObjects(1).Height = 288
```



# HeightPercent Property

Returns or sets the height of a 3-D chart as a percentage of the chart width (between 5 and 500 percent). Read/write **Long**.

## Example

This example sets the height of Chart1 to 80 percent of its width. The example should be run on a 3-D chart.

```
Charts("Chart1").HeightPercent = 80
```



[Show All](#)

# Hidden Property

[Hidden property as it applies to the \*\*Range\*\* object.](#)

**True** if the rows or columns are hidden. The specified range must span an entire column or row. Read/write **Variant**.

*expression*.**Hidden**

*expression* Required. An expression that returns one of the above objects.

[Hidden property as it applies to the \*\*Scenario\*\* object.](#)

**True** if the scenario is hidden. The default value is **False**. Read/write **Boolean**.

*expression*.**Hidden**

*expression* Required. An expression that returns one of the above objects.

[Hidden property as it applies to the \*\*TreeviewControl\*\* object.](#)

Returns or sets the hidden status of the [cube](#) field members in the hierarchical member selection control of a cube field. Read/write **Variant**.

*expression*.**Hidden**

*expression* Required. An expression that returns one of the above objects.

## Remarks

Don't confuse this property with the [FormulaHidden](#) property.

**TreeviewControl** object: The **Hidden** property returns or sets an array. Each element of the array corresponds to a [level](#) of the cube field that is hidden. The maximum number of elements is the number of levels in the cube field. Each element of the array is an array of type **String**, containing unique member names that are hidden at the corresponding level of the control. See the [DrilledDown](#) property of the [PivotItem](#) object to determine when members are visible (expanded) in the control.

## Example

This example hides column C on Sheet1.

```
Worksheets("Sheet1").Columns("C").Hidden = True
```

This example hides the second [level](#) member [state].[states].[CA].[Covelo] of the first [cube](#) field in the first PivotTable report.

```
ActiveSheet.PivotTables("PivotTable1").CubeFields(1) _  
    .TreeViewControl.Hidden = _  
        Array(Array(""), Array(""), _  
            Array("[state].[states].[CA].[Covelo]"))
```



[Show All](#)

# HiddenFields Property

Returns an object that represents either a single PivotTable field (a [PivotField](#) object) or a collection of all the fields (a [PivotFields](#) object) that are currently not shown as row, column, page, or data fields. Read-only.

*expression*.**HiddenFields**(*Index*)

*expression* Required. An expression that returns a **PivotTable** object.

**Index** Optional **Variant**. The name or number of the field to be returned (can be an array to specify more than one field).

## Remarks

For [OLAP](#) data sources, this property always returns an empty collection.

## Example

This example adds the hidden field names to a list on a new worksheet.

```
Set nwSheet = Worksheets.Add
nwSheet.Activate
Set pvtTable = Worksheets("Sheet2").Range("A1").PivotTable
rw = 0
For Each pvtField In pvtTable.HiddenFields
    rw = rw + 1
    nwSheet.Cells(rw, 1).Value = pvtField.Name
Next pvtField
```



[Show All](#)

# HiddenItems Property

Returns an object that represents either a single hidden PivotTable item (a [PivotItem](#) object) or a collection of all the hidden items (a [PivotItems](#) object) in the specified field. Read-only.

*expression*.**HiddenItems**(*Index*)

*expression* Required. An expression that returns a **PivotField** object.

**Index** Optional **Variant**. The number or name of the item to be returned (can be an array to specify more than one item).

## Remarks

For [OLAP](#) data sources, this property always returns an empty collection.

## Example

This example adds the names of all the hidden items in the field named "product" to a list on a new worksheet.

```
Set nwSheet = Worksheets.Add  
nwSheet.Activate  
Set pvtTable = Worksheets("Sheet2").Range("A1").PivotTable  
rw = 0  
For Each pvtItem In pvtTable.PivotFields("product").HiddenItems  
    rw = rw + 1  
    nwSheet.Cells(rw, 1).Value = pvtItem.Name  
Next pvtItem
```



[Show All](#)

# HiddenItemsList Property

Returns or sets a **Variant** specifying an array of strings that are hidden items for a PivotTable field. Read/write.

*expression*.**HiddenItemsList**

*expression* Required. An expression that returns a [PivotField](#) object.

## Remarks

The **HiddenItemsList** property is only valid for [Online Analytical Processing \(OLAP\)](#) data sources; using this property on non-OLAP data sources will return a run-time error.

## Example

The example sets the item list so that only certain items are displayed. It assumes an OLAP PivotTable exists on the active worksheet.

```
Sub UseHiddenItemsList()  
    ActiveSheet.PivotTables(1).PivotFields(1).HiddenItemsList = _  
        Array("[Product].[All Products].[Food]", _  
            "[Product].[All Products].[Drink]")  
End Sub
```



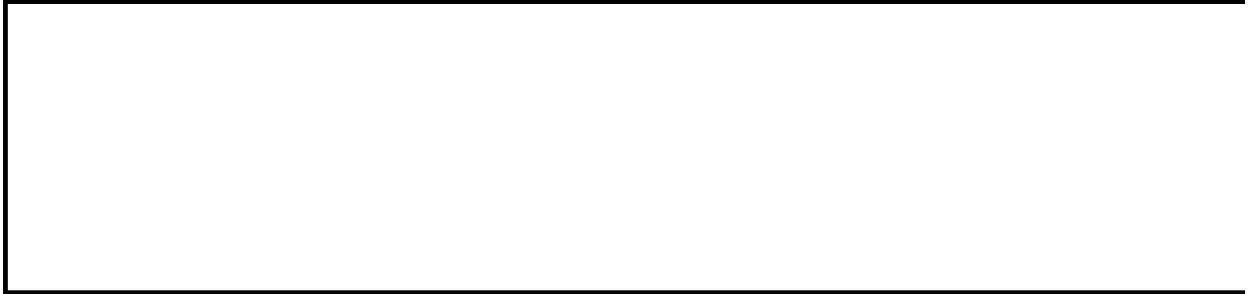
# HighlightChangesOnScreen Property

**True** if changes to the shared workbook are highlighted on-screen. Read/write **Boolean**.

## Example

This example highlights changes to the shared workbook.

`ThisWorkbook.HighlightChangesOnScreen`



# HiLoLines Property

Returns a [HiLoLines](#) object that represents the high-low lines for a series on a line chart. Applies only to line charts. Read-only.

## Example

This example turns on high-low lines for chart group one in Chart1 and then sets their line style, weight, and color. The example should be run on a 2-D line chart that has three series of stock-quote-like data (high-low-close).

```
With Charts("Chart1").ChartGroups(1)
    .HasHiLoLines = True
    With .HiLoLines.Border
        .LineStyle = xlThin
        .Weight = xlMedium
        .ColorIndex = 3
    End With
End With
```



# Hinstance Property

Returns the instance handle of the instance that is calling Microsoft Excel. Read-only **Long**.

*expression*.**Hinstance**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel notifies the user about the instance handle of the instance that is calling Excel.

```
Sub CheckHinstance()  
    MsgBox Application.Hinstance  
End Sub
```



[Show All](#)

# HorizontalAlignment Property

 [HorizontalAlignment](#) property as it applies to the [Style](#) and [TextFrame](#) objects.

Returns or sets the horizontal alignment for the specified object. For all objects, this can be one of the following **XIHAAlign** constants. Read/write [XIHAAlign](#).

XIHAAlign can be one of these XIHAAlign constants.

**xIHAAlignCenter**

**xIHAAlignCenterAcrossSelection**

**xIHAAlignDistributed**

**xIHAAlignFill**

**xIHAAlignGeneral**

**xIHAAlignJustify**

**xIHAAlignLeft**

**xIHAAlignRight**

*expression*.**HorizontalAlignment**

*expression* Required. An expression that returns one of the above objects.

 [HorizontalAlignment](#) property as it applies to the [AxisTitle](#), [CellFormat](#), [ChartTitle](#), [DataLabel](#), [DataLabels](#), [DisplayUnitLabel](#), and [Range](#) objects.

Returns or sets the horizontal alignment for the specified object. Read/write **Variant**.

*expression*.**HorizontalAlignment**

*expression* Required. An expression that returns one of the above objects.

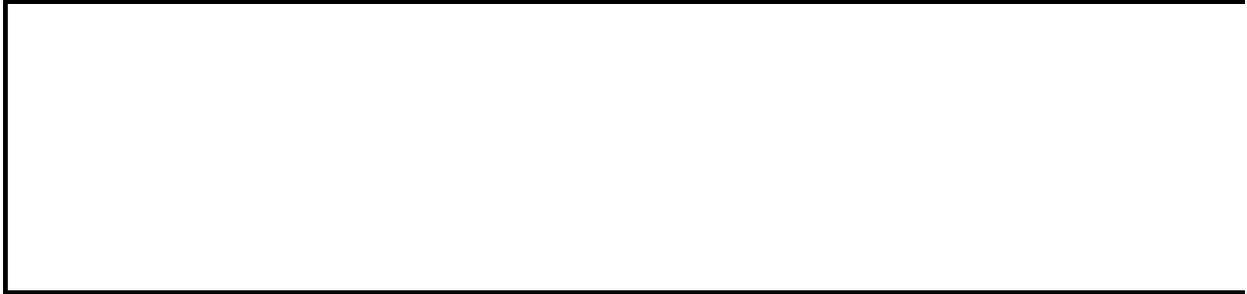
## Remarks

Some of these constants may not be available to you, depending on the language support (U.S. English, for example) that you've selected or installed.

## Example

This example left aligns the range A1:A5 on Sheet1.

```
Worksheets("Sheet1").Range("A1:A5").HorizontalAlignment = xlLeft
```



[Show All](#)

# HorizontalFlip Property

**True** if the specified shape is flipped around the horizontal axis. Read-only [MsoTriState](#).

MsoTriState can be one of these MsoTriState constants.

**msoCTrue**

**msoFalse**

**msoTriStateMixed**

**msoTriStateToggle**

**msoTrue** The specified shape is flipped around the horizontal axis.

*expression*.**HorizontalFlip**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example restores each shape on myDocument to its original state if it's been flipped horizontally or vertically.

```
Set myDocument = Worksheets(1)
For Each s In myDocument.Shapes
    If s.HorizontalFlip Then s.Flip msoFlipHorizontal
    If s.VerticalFlip Then s.Flip msoFlipVertical
Next
```



# HPageBreaks Property

Returns an [HPageBreaks](#) collection that represents the horizontal page breaks on the sheet. Read-only.

For information about returning a single member of a collection, see [Returning an Object from a Collection](#).

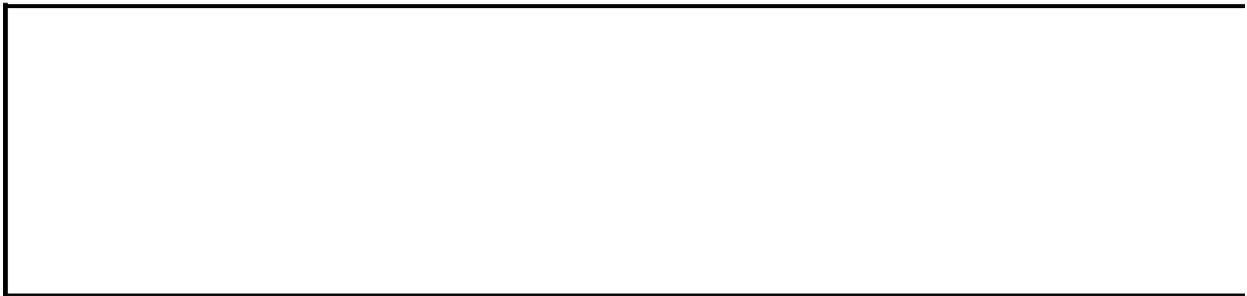
## Remarks

**Note** There is a limit of 1026 horizontal page breaks per sheet.

## Example

This example displays the number of full-screen and print-area horizontal page breaks.

```
For Each pb in Worksheets(1).HPageBreaks
    If pb.Extent = xlPageBreakFull Then
        cFull = cFull + 1
    Else
        cPartial = cPartial + 1
    End If
Next
MsgBox cFull & " full-screen page breaks, " & cPartial & _
    " print-area page breaks"
```



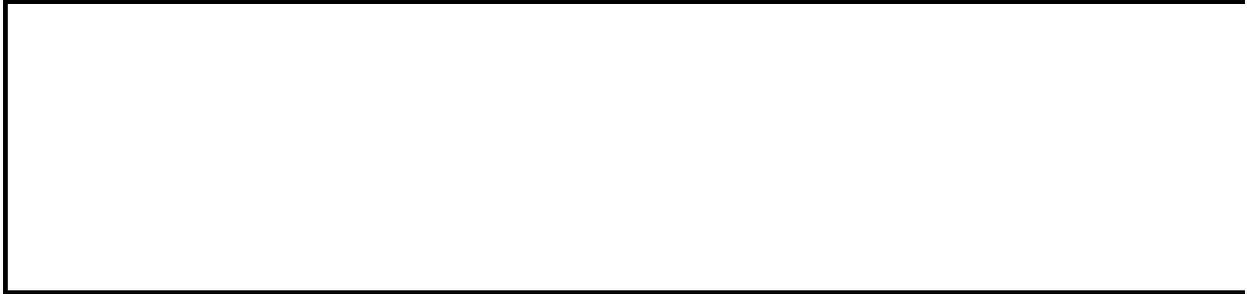
# HTMLProject Property

Returns the [HTMLProject](#) object in the specified workbook, which represents a top-level project branch, as in the Project Explorer in the Microsoft Script Editor. Read-only.

## Example

This example refreshes the HTML project in the active workbook.

```
ActiveWorkbook.HTMLProject.RefreshProject
```



[Show All](#)

# HtmlType Property

Returns or sets the type of HTML generated by Microsoft Excel when you save the specified item to a Web page. Can be one of the **XlHtmlType** constants listed in the following table, specifying whether the item is static or interactive in the Web page. The default value is **xlHtmlStatic**. Read/write [XlHtmlType](#).

XlHtmlType can be one of these XlHtmlType constants.

**xlHtmlCalc**. Use the Spreadsheet component.

**xlHtmlChart**. Use the Chart component.

**xlHtmlList**. Use the PivotTable component.

**xlHtmlStatic**. Use static (noninteractive) HTML for viewing only.

*expression*.**HtmlType**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example saves the range D5:D9 on the First Quarter worksheet in the active workbook to a Web page called “stockreport.htm.” You use the Spreadsheet component to add interactivity to the Web page.

```
ActiveWorkbook.PublishObjects.Add( _  
    SourceType:=xlSourceRange, _  
    Filename:="\\Server2\Q1\stockreport.htm", _  
    Sheet:="First Quarter", _  
    Source:="D5:D9", _  
    HtmlType:=xlHTMLCalc).Publish
```



# Hwnd Property

Returns a **Long** indicating the top-level window handle of the Microsoft Excel window. Read-only.

*expression*.**Hwnd**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel notifies the user of the top-level window handle of the Excel window.

```
Sub CheckHwnd()  
    MsgBox "The top-level window handle is: " & _  
        Application.Hwnd  
End Sub
```



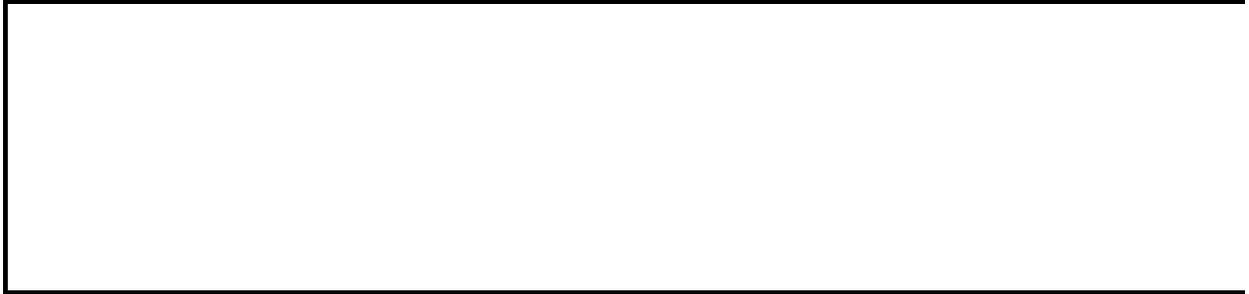
# Hyperlink Property

Returns a [Hyperlink](#) object that represents the hyperlink for the shape.

## Example

This example loads the document attached to the hyperlink on shape one.

```
Worksheets(1).Shapes(1).Hyperlink.Follow NewWindow:=True
```



# Hyperlinks Property

Returns a [Hyperlinks](#) collection that represents the hyperlinks for the range or worksheet.

For more information about returning an object from a collection, see [Returning an Object from a Collection](#).

## Example

This example checks to see whether any of the hyperlinks on worksheet one contain the word “Microsoft.”

```
For Each h in Worksheets(1).Hyperlinks
    If Instr(h.Name, "Microsoft") <> 0 Then h.Follow
Next
```



[Show All](#)

# ID Property

 [ID property as it applies to the \*\*Shape\*\* and \*\*ShapeRange\*\* objects.](#)

Returns the type for the specified object. Read-only **Long**.

*expression.ID*

*expression* Required. An expression that returns one of the above objects.

 [ID property as it applies to the \*\*Range\*\* object.](#)

Returns or sets the identifying label for the specified cell when the page is saved as a Web page. Read/write **String**.

*expression.ID*

*expression* Required. An expression that returns a **Range** object.

## Remarks

You can use an ID label as a hyperlink reference in other HTML documents or on the same Web page.

## Example

This example sets the ID of cell A1 on the active worksheet to "target".

```
ActiveSheet.Range("A1").ID = "target"
```

Later, the document is saved as a Web page, and the following line of HTML is added to the Web page.

```
<A HREF="#target">Quarterly earnings</A>
```

When the user then views the page in a Web browser and clicks the hyperlink, the browser displays the cell.



# Ignore Property

Allows the user to set or return the state of an error checking option for a range. **False** enables an error checking option for a range. **True** disables an error checking option for a range. Read/write **Boolean**.

*expression*.**Ignore**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Reference the [ErrorCheckingOptions](#) object to view a list of index values associated with error checking options.

## Example

This example disables the ignore flag in cell A1 for checking empty cell references.

```
Sub IgnoreChecking()  
    Range("A1").Select  
  
    ' Determine if empty cell references error checking is on, if no  
    If Application.Range("A1").Errors(xlEmptyCellReferences).Ignore  
        Application.Range("A1").Errors(xlEmptyCellReferences).Ignore  
        MsgBox "Empty cell references error checking has been enable  
    Else  
        MsgBox "Empty cell references error checking is already enab  
    End If  
  
End Sub
```



# IgnoreBlank Property

**True** if blank values are permitted by the range data validation. Read/write **Boolean**.

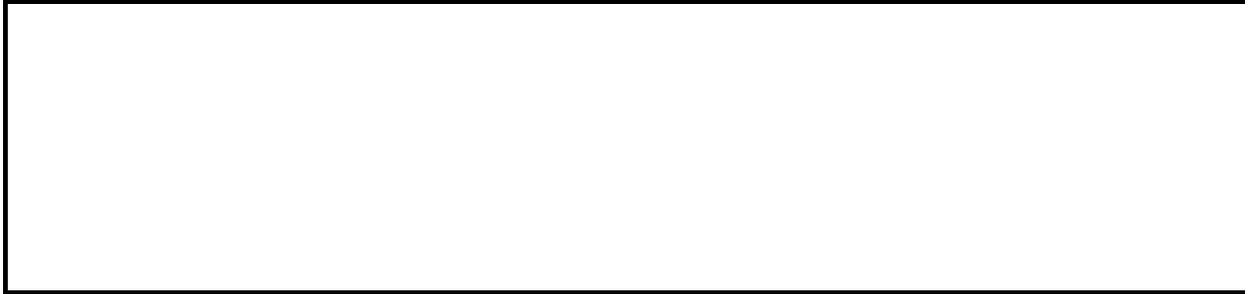
## Remarks

If the **IgnoreBlank** property is **True**, cell data is considered valid if the cell is blank, or if a cell referenced by either the **MinVal** or **MaxVal** property is blank.

## Example

This example causes data validation for cell E5 to allow blank values.

```
Range("e5").Validation.IgnoreBlank = True
```



# IgnoreCaps Property

**False** instructs Microsoft Excel to check for uppercase words, **True** instructs Excel to ignore words in uppercase when using the spelling checker. Read/write **Boolean**.

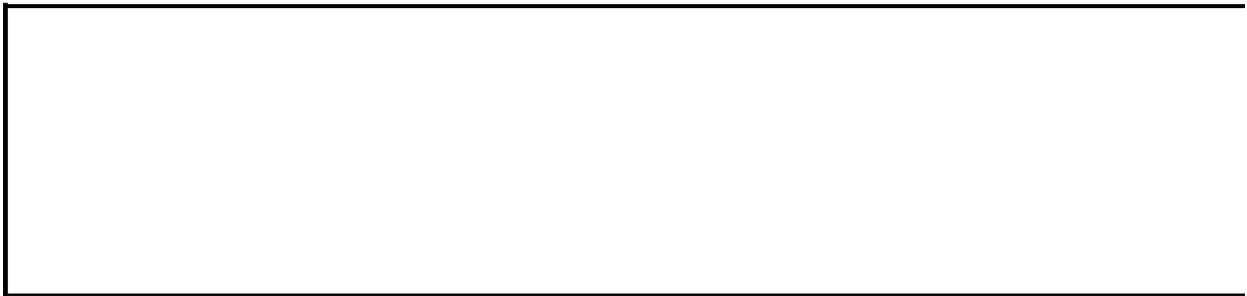
*expression.IgnoreCaps*

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel determines what the setting is for checking the spelling of uppercase words and notifies the user.

```
Sub SpellingOptionsCheck()  
    If Application.SpellingOptions.IgnoreCaps = True Then  
        MsgBox "Spelling options for checking uppercase words is dis  
    Else  
        MsgBox "Spelling options for checking uppercase words is ena  
    End If  
End Sub
```



# IgnoreFileNames Property

**False** instructs Microsoft Excel to check for Internet and file addresses, **True** instructs Excel to ignore Internet and file addresses when using the spell checker.  
Read/write **Boolean**.

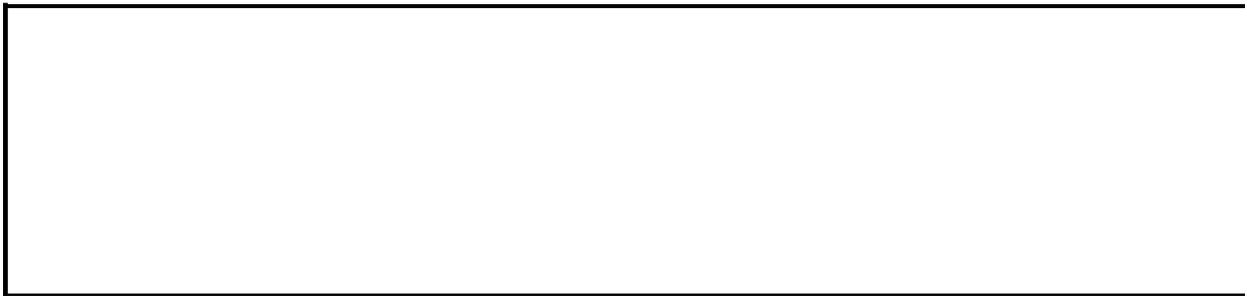
*expression*.**IgnoreFileNames**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel determines what the setting is for checking spelling of Internet and file addresses and notifies the user.

```
Sub SpellingOptionsCheck()  
    If Application.SpellingOptions.IgnoreFileNames = True Then  
        MsgBox "Spelling options for checking Internet and file addr  
    Else  
        MsgBox "Spelling options for checking Internet and file addr  
    End If  
End Sub
```



# IgnoreMixedDigits Property

**False** instructs Microsoft Excel to check for mixed digits, **True** instructs Excel to ignore mixed digits when checking spelling. Read/write **Boolean**.

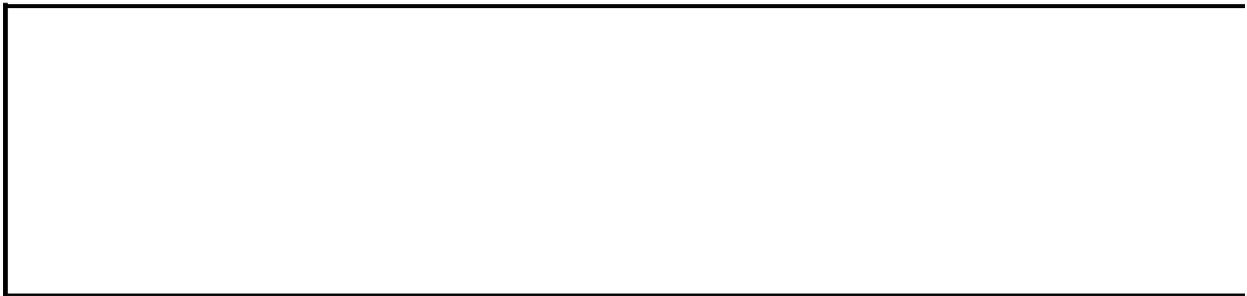
*expression*.**IgnoreMixedDigits**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel determines what the setting is for the checking of spelling for mixed digits and notifies the user.

```
Sub SpellingOptionsCheck()  
    If Application.SpellingOptions.IgnoreMixedDigits = True Then  
        MsgBox "Spelling options for checking mixed digits is disabl  
    Else  
        MsgBox "Spelling options for checking mixed digits is enable  
    End If  
End Sub
```



# IgnoreRemoteRequests Property

**True** if remote DDE requests are ignored. Read/write **Boolean**.

## Example

This example sets the **IgnoreRemoteRequests** property to **True** so that remote DDE requests are ignored.

```
Application.IgnoreRemoteRequests = True
```



# IMEMode Property

Returns or sets the description of the Japanese input rules. Can be one of the **XIIMEMode** constants listed in the following table. Read/write **Long**.

*expression*.**IMEMode**

*expression* Required. An expression that returns one of the objects in the Applies To list.

<b>Constant</b>	<b>Description</b>
<b>xIIMEModeAlpha</b>	Half-width alphanumeric
<b>xIIMEModeAlphaFull</b>	Full-width alphanumeric
<b>xIIMEModeDisable</b>	Disable
<b>xIIMEModeHiragana</b>	Hiragana
<b>xIIMEModeKatakana</b>	Katakana
<b>xIIMEModeKatakanaHalf</b>	Katakana (half-width)
<b>xIIMEModeNoControl</b>	No control
<b>xIIMEModeOff</b>	Off (English mode)
<b>xIIMEModeOn</b>	On

## Remarks

Note that this property can be set only when Japanese language support has been installed and selected.

## Example

This example sets the data input rule for cell E5.

```
With Range("E5").Validation
    .Add Type:=xlValidateWholeNumber, _
        AlertStyle:= xlValidAlertStop, _
        Operator:=xlBetween, Formula1:="5", Formula2:="10"
    .InputTitle = "整数値"
    .ErrorTitle = "整数値"
    .InputMessage = "5 から 10 の整数を入力してください。"
    .ErrorMessage = "入力できるのは 5 から 10 までの値です。"
    .IMEMode = xlIMEModeAlpha
End With
```



[Show All](#)

# InactiveListBorderVisible Property

A **Boolean** value that specifies whether [list](#) borders are visible when a list is not active. Returns **True** if the border is visible. Read/write **Boolean**.

*expression*.**InactiveListBorderVisible**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## **Remarks**

Setting this property will affect all the lists that are on the worksheet.

## Example

The following example hides the borders of inactive lists in the workbook.

```
Sub HideListBorders()
```

```
    ActiveWorkbook.InactiveListBorderVisible = False
```

```
End Sub
```



# InCellDropdown Property

**True** if data validation displays a drop-down list that contains acceptable values.  
Read/write **Boolean**.

## Remarks

This property is ignored if the validation type isn't **xlValidateList**.

Use the *Minimum* argument of the **Add** or **Modify** method of the **Validation** object to specify the range that contains valid data.

## Example

This example adds data validation to cell E5. The range A1:A10 contains the acceptable values for the cell and the cell displays a drop-down list that contains those values.

```
With Range("e5").Validation
    .Add xlValidateList, xlValidAlertStop, xlBetween, "=$A$1:$A$10"
    .InCellDropdown = True
End With
```



# IncludeAlignment Property

**True** if the style includes the **AddIndent**, **HorizontalAlignment**, **VerticalAlignment**, **WrapText**, and **Orientation** properties. Read/write **Boolean**.

## Example

This example sets the style attached to cell A1 on Sheet1 to include alignment format.

```
Worksheets("Sheet1").Range("A1").Style.IncludeAlignment = True
```



# IncludeBorder Property

**True** if the style includes the **Color**, **ColorIndex**, **LineStyle**, and **Weight** border properties. Read/write **Boolean**.

## Example

This example sets the style attached to cell A1 on Sheet1 to include border format.

```
Worksheets("Sheet1").Range("A1").Style.IncludeBorder = True
```



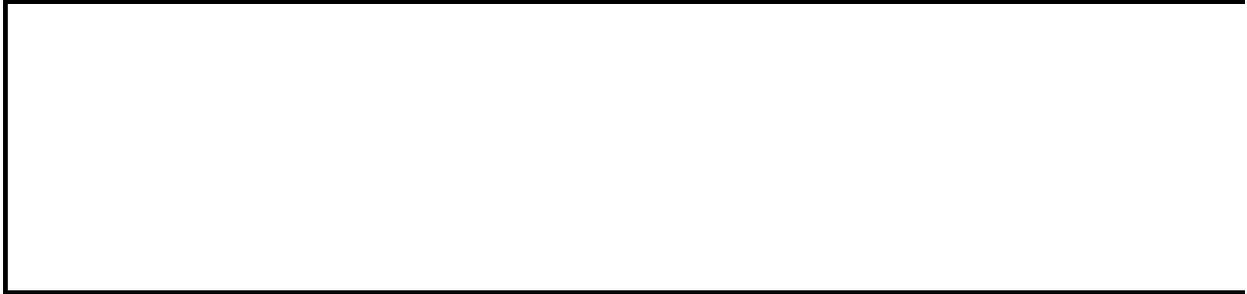
# IncludeFont Property

**True** if the style includes the **Background**, **Bold**, **Color**, **ColorIndex**, **FontStyle**, **Italic**, **Name**, **OutlineFont**, **Shadow**, **Size**, **Strikethrough**, **Subscript**, **Superscript**, and **Underline** font properties. Read/write **Boolean**.

## Example

This example sets the style attached to cell A1 on Sheet1 to include font format.

```
Worksheets("Sheet1").Range("A1").Style.IncludeFont = True
```



# IncludeNumber Property

**True** if the style includes the **NumberFormat** property. Read/write **Boolean**

## Example

This example sets the style attached to cell A1 on Sheet1 to include number format.

```
Worksheets("Sheet1").Range("A1").Style.IncludeNumber = True
```



# IncludePatterns Property

**True** if the style includes the **Color**, **ColorIndex**, **InvertIfNegative**, **Pattern**, **PatternColor**, and **PatternColorIndex** interior properties. Read/write **Boolean**.

## Example

This example sets the style attached to cell A1 on Sheet1 to include pattern format.

```
Worksheets("Sheet1").Range("A1").Style.IncludePatterns = True
```



# IncludeProtection Property

**True** if the style includes the **FormulaHidden** and **Locked** protection properties. Read/write **Boolean**.

## Example

This example sets the style attached to cell A1 on Sheet1 to include protection format.

```
Worksheets("Sheet1").Range("A1").Style.IncludeProtection = True
```



# InconsistentFormula Property

When set to **True** (default), Microsoft Excel identifies cells containing an inconsistent formula in a region. **False** disables the inconsistent formula check. Read/write **Boolean**.

*expression*.**InconsistentFormula**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Consistent formulas in the region must reside to the left and right or above and below the cell containing the inconsistent formula for the **InconsistentFormula** property to work properly.

## Example

In the following example, when the user selects cell B4 (which contains an inconsistent formula), the **AutoCorrect Options** button appears.

```
Sub CheckFormula()  
    Application.ErrorCheckingOptions.InconsistentFormula = True  
  
    Range("A1:A3").Value = 1  
    Range("B1:B3").Value = 2  
    Range("C1:C3").Value = 3  
  
    Range("A4").Formula = "=SUM(A1:A3)" ' Consistent formula.  
    Range("B4").Formula = "=SUM(B1:B2)" ' Inconsistent formula.  
    Range("C4").Formula = "=SUM(C1:C3)" ' Consistent formula.  
  
End Sub
```



[Show All](#)

# IndentLevel Property

 [IndentLevel property as it applies to the \*\*Style\*\* object.](#)

Returns or sets the indent level for the style. Read/write **Long**.

*expression*.**IndentLevel**

*expression* Required. An expression that returns a **Style** object.

 [IndentLevel property as it applies to the \*\*CellFormat\*\* and \*\*Range\*\* objects.](#)

Returns or sets the indent level for the cell or range. Can be an integer from 0 to 15. Read/write **Variant**.

*expression*.**IndentLevel**

*expression* Required. An expression that returns one of the above objects.

## Remarks

Using this property to set the indent level to a number less than 0 (zero) or greater than 15 causes an error.

## Example

[As it applies to the \*\*CellFormat\*\* and \*\*Range\*\* objects.](#)

This example increases the indent level to 15 in cell A10.

```
With Range("A10")  
    .IndentLevel = 15  
End With
```



[Show All](#)

# Index Property

[Index property as it applies to the \*\*ListColumn\*\* object.](#)

Returns the index number of the **ListColumn** object within the **ListColumns** collection. Read-only **Integer**.

*expression*.**Index**

*expression* Required. An expression that returns **ListColumn** object.

[Index property as it applies to the \*\*ListRow\*\* object.](#)

For the **ListRow** object, returns or sets the index number of the object within the **ListRows** collection. Read-only **Long**.

*expression*.**Index**

*expression* Required. An expression that returns **ListRow** object.

[Index property as it applies to the \*\*PivotFormula\*\* object.](#)

For the **PivotFormula** object, returns or sets the index number of the object within the **PivotFormulas** collection. Read/write **Long**.

*expression*.**Index**

*expression* Required. An expression that returns one of the above objects.

[Index property as it applies to all other objects in the \*\*Applies To\*\* list.](#)

Returns the index number of the object within the collection of similar objects. Read-only **Long**.

*expression*.**Index**

*expression* Required. An expression that returns one of the above objects.

For information about using the **Index** worksheet function in Visual Basic, see [Using Worksheet Functions in Visual Basic](#).

## Example

This example displays the tab number of the sheet specified by the name that you type. For example, if Sheet4 is the third tab in the active workbook, the example displays "3" in a message box.

```
Sub DisplayTabNumber()  
    Dim strSheetName as String  
  
    strSheetName = InputBox("Type a sheet name, such as Sheet4.")  
  
    MsgBox "This sheet is tab number " & Sheets(strSheetName).Index  
End Sub
```



[Show All](#)

# IndicatorColorIndex Property

Returns or sets the color of the indicator for error checking options. Read/write [xlColorIndex](#).

XlColorIndex can be one of these XlColorIndex constants.

**xlColorIndexAutomatic** The default system color.

**xlColorIndexNone** Not used with this property.

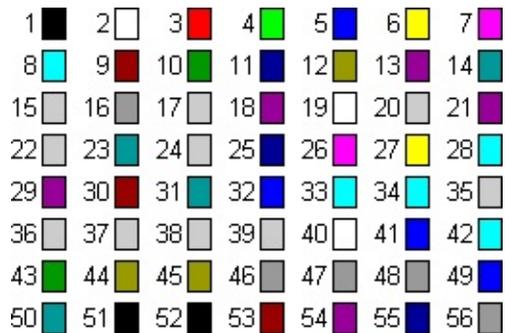
*expression*.**IndicatorColorIndex**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

You can specify a particular color for the indicator by entering the corresponding index value. You can use the **Colors** property to return the current color palette.

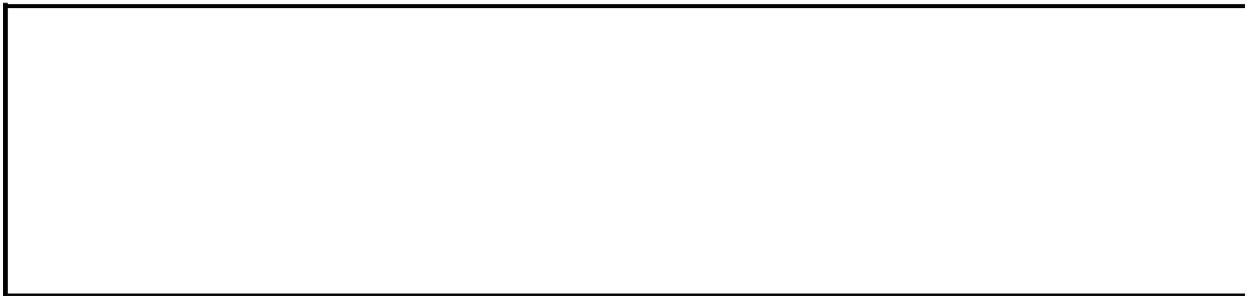
The following illustration shows the color-index values in the default color palette.



## Example

In the following example, Microsoft Excel checks to see if the indicator color for error checking is set to the default system color and notifies the user accordingly.

```
Sub CheckIndexColor()  
    If Application.ErrorCheckingOptions.IndicatorColorIndex = xlColo  
        MsgBox "Your indicator color for error checking is set to th  
    Else  
        MsgBox "Your indicator color for error checking is not set t  
    End If  
End Sub
```



[Show All](#)

# InnerDetail Property

Returns or sets the name of the field that will be shown as detail when the **ShowDetail** property is **True** for the innermost row or column field. Read/write **String**.

## Remarks

This property isn't available for [OLAP](#) data sources.

## Example

This example displays the name of the field that will be shown as detail when the **ShowDetail** property is **True** for the innermost row field or column field.

```
Set pvtTable = Worksheets("Sheet1").Range("A3").PivotTable  
MsgBox pvtTable.InnerDetail
```



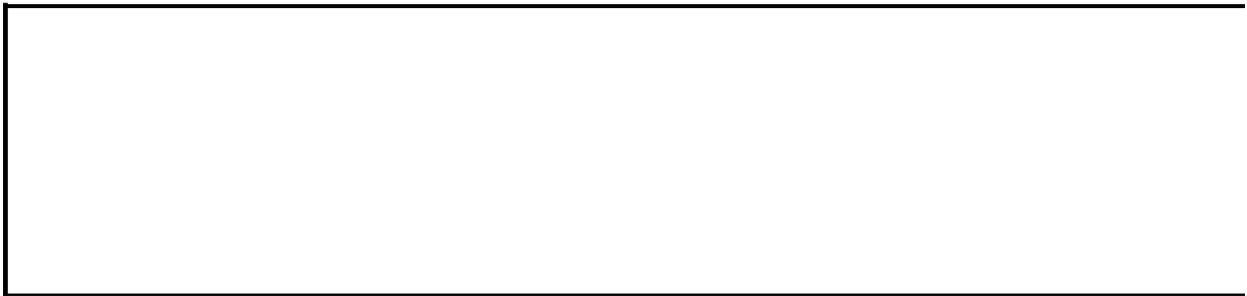
# InputMessage Property

Returns or sets the data validation input message. Read/write **String**.

## Example

This example adds data validation to cell E5 and specifies both the input and error messages.

```
With Range("e5").Validation
    .Add Type:=xlValidateWholeNumber, _
        AlertStyle:= xlValidAlertStop, _
        Operator:=xlBetween, Formula1:="5", Formula2:="10"
    .InputTitle = "Integers"
    .ErrorTitle = "Integers"
    .InputMessage = "Enter an integer from five to ten"
    .ErrorMessage = "You must enter a number from five to ten"
End With
```



# InputTitle Property

Returns or sets the title of the data-validation input dialog box. Read/write **String**.

## Example

This example turns on data validation for cell E5.

```
With Range("e5").Validation
    .Add xlValidateWholeNumber, _
        xlValidAlertInformation, xlBetween, "5", "10"
    .InputTitle = "Integers"
    .ErrorTitle = "Integers"
    .InputMessage = "Enter an integer from five to ten"
    .ErrorMessage = "You must enter a number from five to ten"
End With
```



[Show All](#)

# InsertRowRange Property

Returns a **Range** object representing the [Insert row](#), if any, of a specified **ListObject** object. Read-only **Range**.

*expression*.**InsertRowRange**

*expression* Required. An expression that returns a **ListObject** object.

## Remarks

If the Insert row is not visible because the [list](#) is inactive, the **Nothing** object will be returned.

## Example

The following example activates the range specified by the **InsertRowRange** in the default **ListObject** object in the first worksheet of the active workbook.

```
Function ActivateInsertRow() As Boolean

    Dim wrksht As Worksheet
    Dim objList As ListObject
    Dim objListRng As Range

    Set wrksht = ActiveWorkbook.Worksheets(1)
    Set objList = wrksht.ListObjects(1)
    Set objListRng = objList.InsertRowRange

    If objListRng Is Nothing Then
        ActivateInsertRow = False
    Else
        objListRng.Activate
        ActivateInsertRow = True
    End If

End Function
```



[Show All](#)

# InsideHeight Property

Returns the inside height of the plot area, in [points](#). Read-only **Double**.

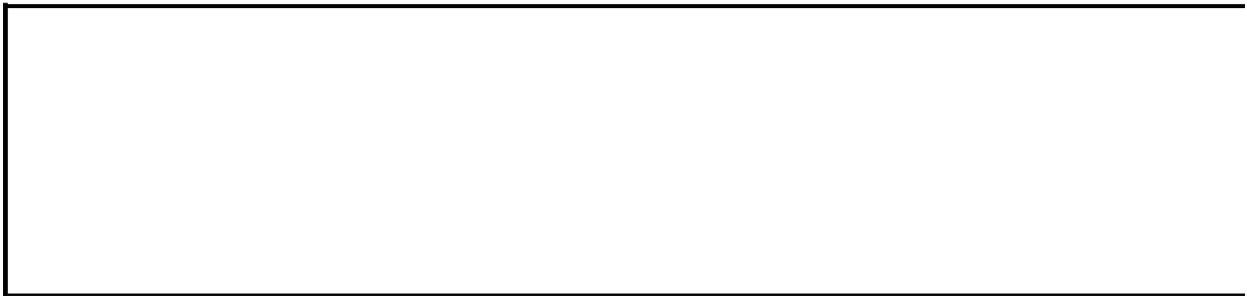
## Remarks

The plot area used for this measurement doesn't include the axis labels. The **Height** property for the plot area uses the bounding rectangle that includes the axis labels.

## Example

This example draws a dotted rectangle around the inside of the plot area in Chart1.

```
With Charts("chart1")
  Set pa = .PlotArea
  With .Shapes.AddShape(msoShapeRectangle, _
    pa.InsideLeft, pa.InsideTop, _
    pa.InsideWidth, pa.InsideHeight)
    .Fill.Transparency = 1
    .Line.DashStyle = msoLineDashDot
  End With
End With
```



[Show All](#)

# InsideLeft Property

Returns the distance from the chart edge to the inside left edge of the plot area, in [points](#). Read-only **Double**.

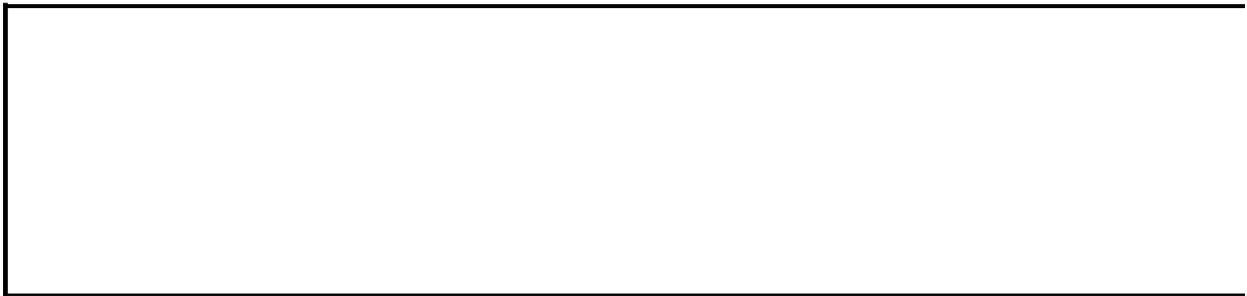
## Remarks

The plot area used for this measurement doesn't include the axis labels. The **Left** property for the plot area uses the bounding rectangle that includes the axis labels.

## Example

This example draws a dotted rectangle around the inside of the plot area in Chart1.

```
With Charts("chart1")
  Set pa = .PlotArea
  With .Shapes.AddShape(msoShapeRectangle, _
    pa.InsideLeft, pa.InsideTop, _
    pa.InsideWidth, pa.InsideHeight)
    .Fill.Transparency = 1
    .Line.DashStyle = msoLineDashDot
  End With
End With
```



[Show All](#)

# InsideTop Property

Returns the distance from the chart edge to the inside top edge of the plot area, in [points](#). Read-only **Double**.

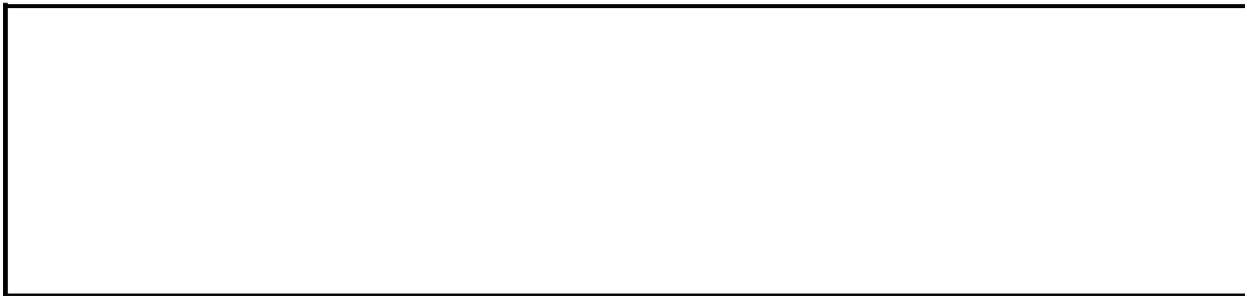
## Remarks

The plot area used for this measurement doesn't include the axis labels. The **Top** property for the plot area uses the bounding rectangle that includes the axis labels.

## Example

This example draws a dotted rectangle around the inside of the plot area in Chart1.

```
With Charts("chart1")
  Set pa = .PlotArea
  With .Shapes.AddShape(msoShapeRectangle, _
    pa.InsideLeft, pa.InsideTop, _
    pa.InsideWidth, pa.InsideHeight)
    .Fill.Transparency = 1
    .Line.DashStyle = msoLineDashDot
  End With
End With
```



[Show All](#)

# InsideWidth Property

Returns the inside width of the plot area, in [points](#). Read-only **Double**.

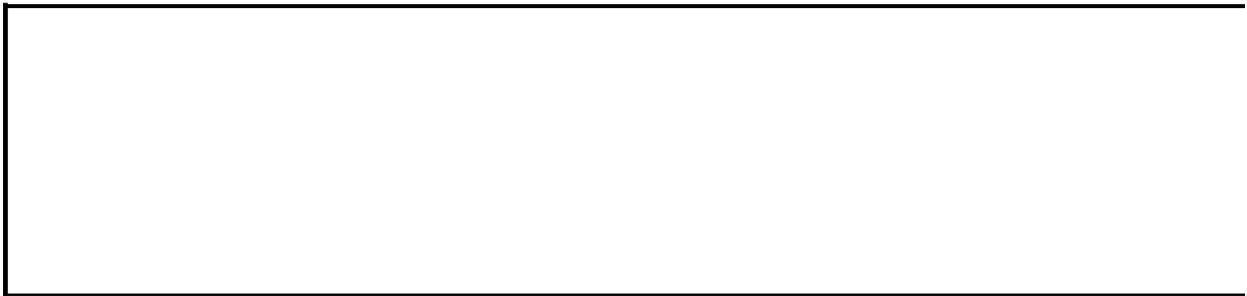
## Remarks

The plot area used for this measurement doesn't include the axis labels. The **Width** property for the plot area uses the bounding rectangle that includes the axis labels.

## Example

This example draws a dotted rectangle around the inside of the plot area in Chart1.

```
With Charts("chart1")
  Set pa = .PlotArea
  With .Shapes.AddShape(msoShapeRectangle, _
    pa.InsideLeft, pa.InsideTop, _
    pa.InsideWidth, pa.InsideHeight)
    .Fill.Transparency = 1
    .Line.DashStyle = msoLineDashDot
  End With
End With
```



# Installed Property

**True** if the add-in is installed. Read/write **Boolean**.

## Remarks

Setting this property to **True** installs the add-in and calls its `Auto_Add` functions.  
Setting this property to **False** removes the add-in and calls its `Auto_Remove` functions.

## Example

This example uses a message box to display the installation status of the Solver add-in.

```
Set a = AddIns("Solver Add-In")
If a.Installed = True Then
    MsgBox "The Solver add-in is installed"
Else
    MsgBox "The Solver add-in is not installed"
End If
```



# Interactive Property

**True** if Microsoft Excel is in interactive mode; this property is usually **True**. If you set the this property to **False**, Microsoft Excel will block all input from the keyboard and mouse (except input to dialog boxes that are displayed by your code). Blocking user input will prevent the user from interfering with the macro as it moves or activates Microsoft Excel objects. Read/write **Boolean**.

## Remarks

This property is useful if you're using DDE or OLE Automation to communicate with Microsoft Excel from another application.

If you set this property to **False**, don't forget to set it back to **True**. Microsoft Excel won't automatically set this property back to **True** when your macro stops running.

## Example

This example sets the **Interactive** property to **False** while it's using DDE in Windows and then sets this property back to **True** when it's finished. This prevents the user from interfering with the macro.

```
Application.Interactive = False
Application.DisplayAlerts = False
channelNumber = Application.DDEInitiate( _
    app:="WinWord", _
    topic:="C:\WINWORD\FORMLETR.DOC")
Application.DDEExecute channelNumber, "[FILEPRINT]"
Application.DDETerminate channelNumber
Application.DisplayAlerts = True
Application.Interactive = True
```



# Intercept Property

Returns or sets the point where the trendline crosses the value axis. Read/write **Double**.

For information about using the **Intercept** worksheet function in Visual Basic, see [Using Worksheet Functions in Visual Basic](#).

## Remarks

Setting this property sets the [InterceptIsAuto](#) property to **False**.

## Example

This example sets trendline one in Chart1 to cross the value axis at 5. The example should be run on a 2-D column chart that contains a single series with a trendline.

```
Charts("Chart1").SeriesCollection(1).Trendlines(1).Intercept = 5
```



# InterceptIsAuto Property

**True** if the point where the trendline crosses the value axis is automatically determined by the regression. Read/write **Boolean**.

## Remarks

Setting the [Intercept](#) property sets this property to **False**.

## Example

This example sets Microsoft Excel to automatically determine the trendline intercept point for Chart1. The example should be run on a 2-D column chart that contains a single series with a trendline.

```
Charts("Chart1").SeriesCollection(1).Trendlines(1) _  
    .InterceptIsAuto = True
```



[Show All](#)

# Interior Property

[Interior property as it applies to the \*\*CellFormat\*\* object.](#)

Returns an **Interior** object allowing the user to set or return the search criteria based on the cell's interior format.

*expression*.**Interior**

*expression* Required. An expression that returns one of the above objects.

[Interior property as it applies to all other objects in the \*\*Applies To\*\* list.](#)

Returns an **Interior** object that represents the interior of the specified object.

*expression*.**Interior**

*expression* Required. An expression that returns one of the above objects.

## Example

 [As it applies to the \*\*CellFormat\*\* object.](#)

This example sets the search criteria to identify cells that contain a solid yellow interior, creates a cell with this condition, finds this cell, and notifies the user.

```
Sub SearchCellFormat()  
  
    ' Set the search criteria for the interior of the cell format.  
    With Application.FindFormat.Interior  
        .ColorIndex = 6  
        .Pattern = xlSolid  
        .PatternColorIndex = xlAutomatic  
    End With  
  
    ' Create a yellow interior for cell A5.  
    Range("A5").Select  
    With Selection.Interior  
        .ColorIndex = 6  
        .Pattern = xlSolid  
        .PatternColorIndex = xlAutomatic  
    End With  
    Range("A1").Select  
    MsgBox "Cell A5 has a yellow interior."  
  
    ' Find the cells based on the search criteria.  
    Cells.Find(What:="", After:=ActiveCell, LookIn:=xlFormulas, Look  
        xlPart, SearchOrder:=xlByRows, SearchDirection:=xlNext, Matc  
        , SearchFormat:=True).Activate  
    MsgBox "Microsoft Excel has found this cell matching the search  
  
End Sub
```

 [As it applies to all other objects in the \*\*Applies To\*\* list.](#)

This example sets the interior color for cell A1 on Sheet1 to cyan.

```
Sub SetColor()  
  
    Worksheets("Sheet1").Range("A1").Interior.ColorIndex = 8 ' Cyan
```

End Sub



# International Property

Returns information about the current country/region and international settings.  
Read-only **Variant**.

*expression*.**International**(*Index*)

## Elements

*expression* Required. An expression that returns an **Application** object.

**Index** Required **Long**. The setting to be returned. Can be one of the **XlApplicationInternational** constants listed in the following tables.

### Brackets and Braces

Index	Type	Meaning
<b>xlLeftBrace</b>	<b>String</b>	Character used instead of the left brace ({} in array literals.
<b>xlLeftBracket</b>	<b>String</b>	Character used instead of the left bracket ([]) in R1C1-style relative references.
<b>xlLowerCaseColumnLetter</b>	<b>String</b>	Lowercase column letter.
<b>xlLowerCaseRowLetter</b>	<b>String</b>	Lowercase row letter.
<b>xlRightBrace</b>	<b>String</b>	Character used instead of the right brace (}) in array literals.
<b>xlRightBracket</b>	<b>String</b>	Character used instead of the right bracket (]) in R1C1-style references.
<b>xlUpperCaseColumnLetter</b>	<b>String</b>	Uppercase column letter.
<b>xlUpperCaseRowLetter</b>	<b>String</b>	Uppercase row letter (for R1C1-style references).

### Country/Region Settings

Index	Type	Meaning
<b>xlCountryCode</b>	<b>Long</b>	Country/Region version of Microsoft Excel.
<b>xlCountrySetting</b>	<b>Long</b>	Current country/region setting in the Windows Control Panel.
<b>xlGeneralFormatName</b>	<b>String</b>	Name of the General number format.

### Currency

Index	Type	Meaning
<b>xlCurrencyBefore</b>	<b>Boolean</b>	<b>True</b> if the currency symbol precedes the currency values; <b>False</b> if it follows them.
<b>xlCurrencyCode</b>	<b>String</b>	Currency symbol.
<b>xlCurrencyDigits</b>	<b>Long</b>	Number of decimal digits to be used in currency formats.
<b>xlCurrencyLeadingZeros</b>	<b>Boolean</b>	<b>True</b> if leading zeros are displayed for zero currency values.
<b>xlCurrencyMinusSign</b>	<b>Boolean</b>	<b>True</b> if you're using a minus sign for negative numbers; <b>False</b> if you're using parentheses.
<b>xlCurrencyNegative</b>	<b>Long</b>	Currency format for negative currency values: 0 = <i>(symbolx)</i> or <i>(xsymbol)</i> 1 = <i>-symbolx</i> or <i>-xsymbol</i> 2 = <i>symbol-x</i> or <i>x-symbol</i> 3 = <i>symbolx-</i> or <i>xsymbol-</i> where <i>symbol</i> is the currency symbol of the country or region. Note that the position of the currency symbol is determined by <b>xlCurrencyBefore</b> .
<b>xlCurrencySpaceBefore</b>	<b>Boolean</b>	<b>True</b> if a space is added before the currency symbol.
<b>xlCurrencyTrailingZeros</b>	<b>Boolean</b>	<b>True</b> if trailing zeros are displayed for zero currency values.
<b>xlNoncurrencyDigits</b>	<b>Long</b>	Number of decimal digits to be used in noncurrency formats.

## Date and Time

Index	Type	Meaning
<b>xl24HourClock</b>	<b>Boolean</b>	<b>True</b> if you're using 24-hour time; <b>False</b> if you're using 12-hour time.
<b>xl4DigitYears</b>	<b>Boolean</b>	<b>True</b> if you're using four-digit years; <b>False</b> if you're using two-digit years. Order of date elements:

<b>xlDateOrder</b>	<b>Long</b>	0 = month-day-year 1 = day-month-year 2 = year-month-day
<b>xlDateSeparator</b>	<b>String</b>	Date separator (/).
<b>xlDayCode</b>	<b>String</b>	Day symbol (d).
<b>xlDayLeadingZero</b>	<b>Boolean</b>	<b>True</b> if a leading zero is displayed in days.
<b>xlHourCode</b>	<b>String</b>	Hour symbol (h).
<b>xlMDY</b>	<b>Boolean</b>	<b>True</b> if the date order is month-day-year for dates displayed in the long form; <b>False</b> if the date order is day-month-year.
<b>xlMinuteCode</b>	<b>String</b>	Minute symbol (m).
<b>xlMonthCode</b>	<b>String</b>	Month symbol (m).
<b>xlMonthLeadingZero</b>	<b>Boolean</b>	<b>True</b> if a leading zero is displayed in months (when months are displayed as numbers). Always returns three characters for backward compatibility. Abbreviated month names are read from Microsoft Windows and can be any length.
<b>xlMonthNameChars</b>	<b>Long</b>	
<b>xlSecondCode</b>	<b>String</b>	Second symbol (s).
<b>xlTimeSeparator</b>	<b>String</b>	Time separator (:).
<b>xlTimeLeadingZero</b>	<b>Boolean</b>	<b>True</b> if a leading zero is displayed in times. Always returns three characters for backward compatibility. Abbreviated weekday names are read from Microsoft Windows and can be any length.
<b>xlWeekdayNameChars</b>	<b>Long</b>	
<b>xlYearCode</b>	<b>String</b>	Year symbol in number formats (y).

## Measurement Systems

Index	Type	Meaning
<b>xlMetric</b>	<b>Boolean</b>	<b>True</b> if you're using the metric system; <b>False</b> if you're using the English measurement system.
<b>xlNonEnglishFunctions</b>	<b>Boolean</b>	<b>True</b> if you're not displaying functions in English.

## Separators

<b>Index</b>	<b>Type</b>	<b>Meaning</b>
<b>xlAlternateArraySeparator</b>	<b>String</b>	Alternate array item separator to be used if the current array separator is the same as the decimal separator.
<b>xlColumnSeparator</b>	<b>String</b>	Character used to separate columns in array literals.
<b>xlDecimalSeparator</b>	<b>String</b>	Decimal separator.
<b>xlListSeparator</b>	<b>String</b>	List separator.
<b>xlRowSeparator</b>	<b>String</b>	Character used to separate rows in array literals.
<b>xlThousandsSeparator</b>	<b>String</b>	Zero or thousands separator.

## Remarks

Symbols, separators, and currency formats shown in the preceding table may differ from those used in your language or geographic location and may not be available to you, depending on the language support (U.S. English, for example) that you've selected or installed.

## Example

This example displays the international decimal separator.

```
MsgBox "The decimal separator is " & _  
Application.International(xlDecimalSeparator)
```



[Show All](#)

# InvalidData Property

Returns a **Boolean** value that indicates if a row in a [list](#) contains cells that don't pass data validation and need to be fixed before any changes can be committed to the Microsoft Windows SharePoint Services site. Read-only **Boolean**.

*expression*.**InvalidData**

*expression* Required. An expression that returns a boolean value.

## Example

The following sample code uses the **InvalidData** property to determine if the first row contains valid data in a list that is linked to a SharePoint site. This sample code requires a list that is connected to a SharePoint site.

```
Sub DisplayDataValidation()  
    Dim wrksht As Worksheet  
  
    Set wrksht = ActiveWorkbook.Worksheets(1)  
  
    If wrksht.ListObjects(1).ListRows(1).InvalidData = True Then  
        MsgBox "There was a problem with the data in the row."  
    Else  
        MsgBox "The row contains valid data."  
    End If  
End Sub
```



# InvertIfNegative Property

**True** if Microsoft Excel inverts the pattern in the item when it corresponds to a negative number. Read/write **Variant** for the **Interior** object, read/write **Boolean** for all other objects.

## Example

This example inverts the pattern for negative values in series one in Chart1. The example should be run on a 2-D column chart.

```
Charts("Chart1").SeriesCollection(1).InvertIfNegative = True
```



# IsAddin Property

**True** if the workbook is running as an add-in. Read/write **Boolean**.

## Remarks

When you set this property to **True**, the workbook has the following characteristics:

- You won't be prompted to save the workbook if changes are made while the workbook is open.
- The workbook window won't be visible.
- Any macros in the workbook won't be visible in the **Macro** dialog box (displayed by pointing to **Macro** on the **Tools** menu and clicking **Macros**).
- Macros in the workbook can still be run from the **Macro** dialog box even though they're not visible. In addition, macro names don't need to be qualified with the workbook name.
- Holding down the SHIFT key when you open the workbook has no effect.

## Example

This example runs a section of code if the workbook is an add-in.

```
If ThisWorkbook.IsAddin Then  
    ' this code runs when the workbook is an add-in  
End If
```



[Show All](#)

# IsCalculated Property

**True** if the PivotTable field or PivotTable item is a calculated field or item.  
Read-only **Boolean**.

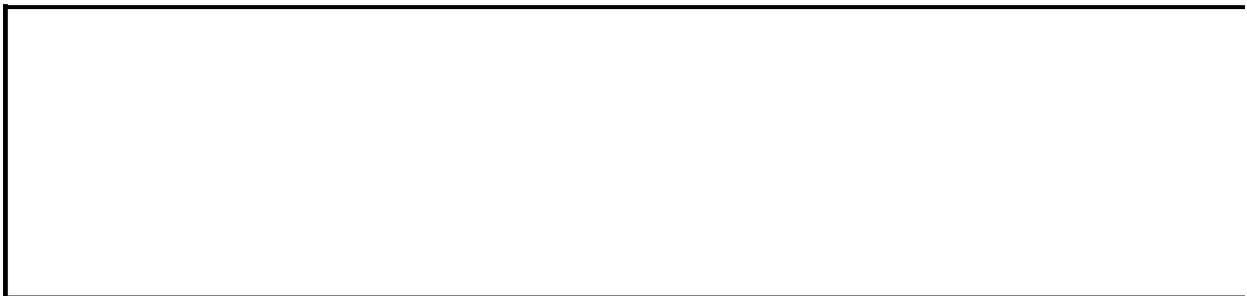
## Remarks

For [OLAP](#) data sources, this property always returns **False**.

## Example

This example disables the **PivotTable Field** dialog box if the specified PivotTable report contains any calculated fields.

```
set pt = Worksheets(1).PivotTables("Pivot1")
For Each fld in pt.PivotFields
    If fld.IsCalculated Then pt.EnableFieldDialog = False
Next
```



# IsConnected Property

Returns **True** if the [MaintainConnection](#) property is **True** and the PivotTable cache is currently connected to its source. Returns **False** if it is not currently connected to its source. Read-only **Boolean**.

*expression*.**IsConnected**

*expression* Required. An expression that returns a [PivotCache](#) object.

## Remarks

The **IsConnected** property does not check to see if the connection is connected. Even if this property returns **True**, sending commands to the provider could result in an error if the connection is no longer valid.

Requires that the cache source type is external and that it is an OLE DB data source.

## Example

The following example determines if the cache is connected to its source and notifies the user. This example assumes a PivotTable exists on the active worksheet.

```
Sub CheckIsConnected()  
  
    ' Handle run-time error if external source is not an OLEDB.  
    On Error GoTo Not_OLEDB  
  
    ' Check connection setting and notify the user accordingly.  
    If Application.ActiveWorkbook.PivotCaches.Item(1).IsConnected =  
        MsgBox "The PivotCache is currently connected to its source."  
    Else  
        MsgBox "The PivotCache is not currently connected to its sou  
    End If  
    Exit Sub  
  
Not_OLEDB:  
    MsgBox "The data source is not an OLEDB data source."  
  
End Sub
```



# IsExportable Property

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Returns **True** if the following conditions are met:

- Microsoft Excel can use the [XPath](#) objects in the specified schema map to export XML data.
- All XML lists mapped to the specified schema map can be exported.

Read-only **Boolean**.

*expression*.**IsExportable**

*expression* Required. An expression that returns one of the objects in the Applies To list.



[Show All](#)

# IsMemberProperty Property

Returns **True** when the PivotField contains member properties. Read-only **Boolean**.

*expression*.**IsMemberProperty**

*expression* Required. An expression that returns a [PivotField](#) object.

## Remarks

This property will return a run-time error if an [Online Analytical Processing \(OLAP\)](#) data source is not used.

## Example

This example determines if the PivotTable field contains member properties and notifies the user. It assumes that a PivotTable exists on the active worksheet and that it is connected to an OLAP data source.

```
Sub CheckForMembers()  
  
    Dim pvtTable As PivotTable  
    Dim pvtField As PivotField  
  
    Set pvtTable = ActiveSheet.PivotTables(1)  
    Set pvtField = pvtTable.PivotFields(1)  
  
    ' Determine if member properties exist and notify user.  
    If pvtField.IsMemberProperty = True Then  
        MsgBox "The PivotField contains member properties."  
    Else  
        MsgBox "The PivotField does not contain member properties."  
    End If  
  
End Sub
```



[Show All](#)

# IsPercent Property

Returns a **Boolean** value. Returns **True** only if the number data for the **ListColumn** object will be shown in percentage formatting. Read-only **Boolean**.

This property is used only for lists that are linked to a Microsoft Windows SharePoint Services site.

*expression*.**IsPercent**

*expression* Required. An expression that returns a **ListDataFormat** object.

## Remarks

In Excel, you cannot set any of the properties associated with the **ListDataFormat** object. You can set these properties, however, by modifying the list on the SharePoint site.

## Example

The following example returns the setting of the **IsPercent** property for the third column of the [list](#) in Sheet1 of the active workbook.

```
Function GetIsPercent() As Boolean
    Dim wrksht As Worksheet
    Dim objListCol As ListColumn

    Set wrksht = ActiveWorkbook.Worksheets("Sheet1")
    Set objListCol = wrksht.ListObjects(1).ListColumns(3)

    GetIsPercent = objListCol.ListDataFormat.IsPercent
End Function
```



# IsValid Property

Returns a Boolean that indicates whether the specified calculated member has been successfully instantiated with the OLAP provider during the current session.

*expression*.**IsValid**

*expression* Required. An expression that returns a [CalculatedMember](#) object.

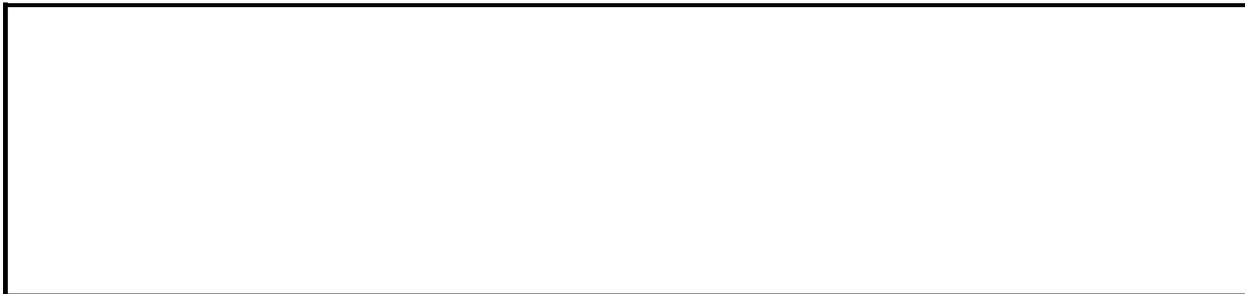
## Remarks

This property returns **True** even if the PivotTable is not connected to its data source. Make sure that the PivotTable is connected before querying the value of the **IsValid** Property.

## Example

This example notifies the user about whether the calculated member is valid or not. It assumes a PivotTable exists on the active worksheet.

```
Sub CheckValidity()  
  
    Dim pvtTable As PivotTable  
    Dim pvtCache As PivotCache  
  
    Set pvtTable = ActiveSheet.PivotTables(1)  
    Set pvtCache = Application.ActiveWorkbook.PivotCaches.Item(1)  
  
    ' Make connection for PivotTable before testing IsValid property  
    pvtCache.MakeConnection  
  
    ' Check if calculated member is valid.  
    If pvtTable.CalculatedMembers.Item(1).IsValid = True Then  
        MsgBox "The calculated member is valid."  
    Else  
        MsgBox "The calculated member is not valid."  
    End If  
  
End Sub
```



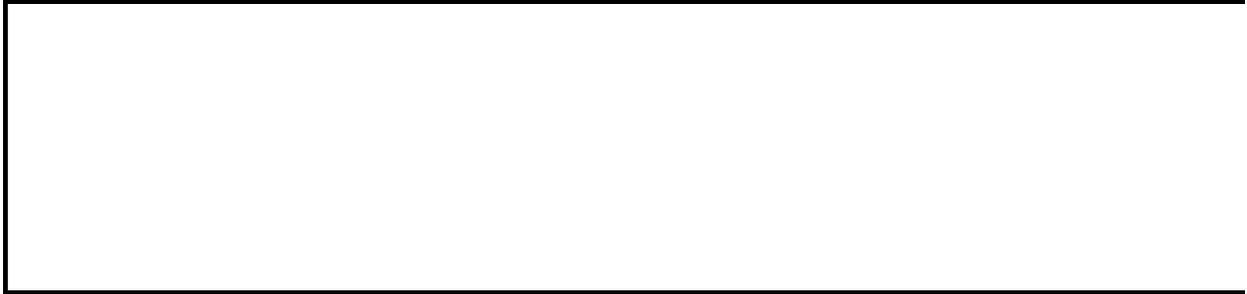
# Italic Property

**True** if the font style is italic. Read/write **Boolean**.

## Example

This example sets the font style to italic for the range A1:A5 on Sheet1.

```
Worksheets("Sheet1").Range("A1:A5").Font.Italic = True
```



[Show All](#)

# Item Property

 [Item property as it applies to the \*\*Adjustments\*\* object.](#)

Returns or sets the adjustment value specified by the **Index** argument. For linear adjustments, an adjustment value of 0.0 generally corresponds to the left or top edge of the shape, and a value of 1.0 generally corresponds to the right or bottom edge of the shape. However, adjustments can pass beyond shape boundaries for some shapes. For radial adjustments, an adjustment value of 1.0 corresponds to the width of the shape. For angular adjustments, the adjustment value is specified in degrees. The **Item** property applies only to shapes that have adjustments. Read/write **Single**.

*expression*.**Item**(**Index**)

*expression* Required. An expression that returns an **Adjustments** object.

**Index** Required **Long**. The index number of the adjustment.

## Remarks

AutoShapes, connectors, and WordArt objects can have up to eight adjustments.

[Item property as it applies to the \*\*Areas\*\*, \*\*Filters\*\*, \*\*HPageBreaks\*\*, \*\*Panes\*\*, \*\*Phonetics\*\*, \*\*RecentFiles\*\*, and \*\*VPageBreaks\*\* objects.](#)

Returns a single object from a collection.

*expression*.**Item**(*Index*)

*expression* Required. An expression that returns one of the above objects.

**Index** Required **Long**. The index number of the object.

[Item property as it applies to the \*\*Borders\*\* object.](#)

Returns a **Border** object that represents one of the borders of either a range of cells or a style.

*expression*.**Item**(*Index*)

*expression* Required. An expression that returns a **Borders** collection.

**Index** Required [XlBordersIndex](#).

XlBordersIndex can be one of these XlBordersIndex constants.

**xlDiagonalDown**

**xlDiagonalUp**

**xlEdgeBottom**

**xlEdgeLeft**

**xlEdgeRight**

**xlEdgeTop**

**xlInsideHorizontal**

**xlInsideVertical**

 [Item](#) property as it applies to the **Dialogs** object.

Returns a **Dialog** object that represents a single built-in dialog box.

*expression*.**Item**(*Index*)

*expression* Required. An expression that returns a **Dialogs** object.

*Index* Required [XlBuiltInDialog](#).

XlBuiltInDialog can be one of these XlBuiltInDialog constants.

**\_xlDialogChartSourceData**

**\_xlDialogPhonetic**

**xlDialogActivate**

**xlDialogActiveCellFont**

**xlDialogAddChartAutoformat**

**xlDialogAddinManager**

**xlDialogAlignment**

**xlDialogApplyNames**

**xlDialogApplyStyle**

**xlDialogAppMove**

**xlDialogAppSize**

**xlDialogArrangeAll**

**xlDialogAssignToObject**

**xlDialogAssignToTool**

**xlDialogAttachText**

**xlDialogAttachToolbars**

**xlDialogAutoCorrect**

**xlDialogAxes**

**xlDialogBorder**

**xlDialogCalculation**

**xlDialogCellProtection**

**xlDialogChangeLink**

**xlDialogChartAddData**

**xlDialogChartLocation**

**xlDialogChartOptionsDataLabelMultiple**  
**xlDialogChartOptionsDataLabels**  
**xlDialogChartOptionsDataTable**  
**xlDialogChartSourceData**  
**xlDialogChartTrend**  
**xlDialogChartType**  
**xlDialogChartWizard**  
**xlDialogCheckboxProperties**  
**xlDialogClear**  
**xlDialogColorPalette**  
**xlDialogColumnWidth**  
**xlDialogCombination**  
**xlDialogConditionalFormatting**  
**xlDialogConsolidate**  
**xlDialogCopyChart**  
**xlDialogCopyPicture**  
**xlDialogCreateNames**  
**xlDialogCreatePublisher**  
**xlDialogCustomizeToolbar**  
**xlDialogCustomViews**  
**xlDialogDataDelete**  
**xlDialogDataLabel**  
**xlDialogDataLabelMultiple**  
**xlDialogDataSeries**  
**xlDialogDataValidation**  
**xlDialogDefineName**  
**xlDialogDefineStyle**  
**xlDialogDeleteFormat**  
**xlDialogDeleteName**  
**xlDialogDemote**  
**xlDialogDisplay**  
**xlDialogEditboxProperties**  
**xlDialogEditColor**

**xlDialogEditDelete**  
**xlDialogEditionOptions**  
**xlDialogEditSeries**  
**xlDialogErrorbarX**  
**xlDialogErrorbarY**  
**xlDialogErrorChecking**  
**xlDialogEvaluateFormula**  
**xlDialogExternalDataProperties**  
**xlDialogExtract**  
**xlDialogFileDelete**  
**xlDialogFileSharing**  
**xlDialogFillGroup**  
**xlDialogFillWorkgroup**  
**xlDialogFilter**  
**xlDialogFilterAdvanced**  
**xlDialogFindFile**  
**xlDialogFont**  
**xlDialogFontProperties**  
**xlDialogFormatAuto**  
**xlDialogFormatChart**  
**xlDialogFormatCharttype**  
**xlDialogFormatFont**  
**xlDialogFormatLegend**  
**xlDialogFormatMain**  
**xlDialogFormatMove**  
**xlDialogFormatNumber**  
**xlDialogFormatOverlay**  
**xlDialogFormatSize**  
**xlDialogFormatText**  
**xlDialogFormulaFind**  
**xlDialogFormulaGoto**  
**xlDialogFormulaReplace**  
**xlDialogFunctionWizard**

**xlDialogGallery3dArea**  
**xlDialogGallery3dBar**  
**xlDialogGallery3dColumn**  
**xlDialogGallery3dLine**  
**xlDialogGallery3dPie**  
**xlDialogGallery3dSurface**  
**xlDialogGalleryArea**  
**xlDialogGalleryBar**  
**xlDialogGalleryColumn**  
**xlDialogGalleryCustom**  
**xlDialogGalleryDoughnut**  
**xlDialogGalleryLine**  
**xlDialogGalleryPie**  
**xlDialogGalleryRadar**  
**xlDialogGalleryScatter**  
**xlDialogGoalSeek**  
**xlDialogGridlines**  
**xlDialogImportTextFile**  
**xlDialogInsert**  
**xlDialogInsertHyperlink**  
**xlDialogInsertNameLabel**  
**xlDialogInsertObject**  
**xlDialogInsertPicture**  
**xlDialogInsertTitle**  
**xlDialogItemProperties**  
**xlDialogLabelProperties**  
**xlDialogListboxProperties**  
**xlDialogMacroOptions**  
**xlDialogMailEditMailer**  
**xlDialogMailLogon**  
**xlDialogMailNextLetter**  
**xlDialogMainChart**  
**xlDialogMainChartType**

**xlDialogMenuEditor**  
**xlDialogMove**  
**xlDialogNew**  
**xlDialogNewWebQuery**  
**xlDialogNote**  
**xlDialogObjectProperties**  
**xlDialogObjectProtection**  
**xlDialogOpen**  
**xlDialogOpenLinks**  
**xlDialogOpenMail**  
**xlDialogOpenText**  
**xlDialogOptionsCalculation**  
**xlDialogOptionsChart**  
**xlDialogOptionsEdit**  
**xlDialogOptionsGeneral**  
**xlDialogOptionsListsAdd**  
**xlDialogOptionsME**  
**xlDialogOptionsTransition**  
**xlDialogOptionsView**  
**xlDialogOutline**  
**xlDialogOverlay**  
**xlDialogOverlayChartType**  
**xlDialogPageSetup**  
**xlDialogParse**  
**xlDialogPasteNames**  
**xlDialogPasteSpecial**  
**xlDialogPatterns**  
**xlDialogPhonetic**  
**xlDialogPivotCalculatedField**  
**xlDialogPivotCalculatedItem**  
**xlDialogPivotClientServerSet**  
**xlDialogPivotFieldGroup**  
**xlDialogPivotFieldProperties**

**xlDialogPivotFieldUngroup**  
**xlDialogPivotShowPages**  
**xlDialogPivotSolveOrder**  
**xlDialogPivotTableOptions**  
**xlDialogPivotTableWizard**  
**xlDialogPlacement**  
**xlDialogPrint**  
**xlDialogPrinterSetup**  
**xlDialogPrintPreview**  
**xlDialogPromote**  
**xlDialogProperties**  
**xlDialogProtectDocument**  
**xlDialogProtectSharing**  
**xlDialogPublishAsWebPage**  
**xlDialogPushbuttonProperties**  
**xlDialogReplaceFont**  
**xlDialogRoutingSlip**  
**xlDialogRowHeight**  
**xlDialogRun**  
**xlDialogSaveAs**  
**xlDialogSaveCopyAs**  
**xlDialogSaveNewObject**  
**xlDialogSaveWorkbook**  
**xlDialogSaveWorkspace**  
**xlDialogScale**  
**xlDialogScenarioAdd**  
**xlDialogScenarioCells**  
**xlDialogScenarioEdit**  
**xlDialogScenarioMerge**  
**xlDialogScenarioSummary**  
**xlDialogScrollbarProperties**  
**xlDialogSearch**  
**xlDialogSelectSpecial**

**xlDialogSendMail**  
**xlDialogSeriesAxes**  
**xlDialogSeriesOptions**  
**xlDialogSeriesOrder**  
**xlDialogSeriesShape**  
**xlDialogSeriesX**  
**xlDialogSeriesY**  
**xlDialogSetBackgroundPicture**  
**xlDialogSetPrintTitles**  
**xlDialogSetUpdateStatus**  
**xlDialogShowDetail**  
**xlDialogShowToolbar**  
**xlDialogSize**  
**xlDialogSort**  
**xlDialogSortSpecial**  
**xlDialogSplit**  
**xlDialogStandardFont**  
**xlDialogStandardWidth**  
**xlDialogStyle**  
**xlDialogSubscribeTo**  
**xlDialogSubtotalCreate**  
**xlDialogSummaryInfo**  
**xlDialogTable**  
**xlDialogTabOrder**  
**xlDialogTextToColumns**  
**xlDialogUnhide**  
**xlDialogUpdateLink**  
**xlDialogVbaInsertFile**  
**xlDialogVbaMakeAddin**  
**xlDialogVbaProcedureDefinition**  
**xlDialogView3d**  
**xlDialogWebOptionsBrowsers**  
**xlDialogWebOptionsEncoding**

**xlDialogWebOptionsFiles**  
**xlDialogWebOptionsFonts**  
**xlDialogWebOptionsGeneral**  
**xlDialogWebOptionsPictures**  
**xlDialogWindowMove**  
**xlDialogWindowSize**  
**xlDialogWorkbookAdd**  
**xlDialogWorkbookCopy**  
**xlDialogWorkbookInsert**  
**xlDialogWorkbookMove**  
**xlDialogWorkbookName**  
**xlDialogWorkbookNew**  
**xlDialogWorkbookOptions**  
**xlDialogWorkbookProtect**  
**xlDialogWorkbookTabSplit**  
**xlDialogWorkbookUnhide**  
**xlDialogWorkgroup**  
**xlDialogWorkspace**  
**xlDialogZoom**

## Remarks

Using the **Item** property of the **Dialogs** collection and the **Show** method, you can display approximately 200 built-in dialog boxes. Each dialog box has a constant assigned to it; these constants all begin with "xlDialog."

For a table of the available constants and their corresponding argument lists, see [Built-In Dialog Box Argument Lists](#).

The **Item** property of the **Dialogs** collection may fail if you try to show a dialog box in an incorrect context. For example, to display the **Data Labels** dialog box (using the Visual Basic expression `Application.Dialogs(xlDialogDataLabel).Show`), the active sheet must be a chart; otherwise, the property fails.

 [Item property as it applies to the \*\*Errors\*\* object.](#)

Returns a single member of the **Error** object.

*expression*.**Item**(*Index*)

*expression* Required. An expression that returns an **Errors** object.

*Index* Required **Variant**. The *Index* can also be one these [constants](#).

**xlEvaluateToError** The cell evaluates to an error value.

**xlTextDate** The cell contains a text date with 2 digit years.

**xlNumberAsText** The cell contains a number stored as text.

**xlInconsistentFormula** The cell contains an inconsistent formula for a region.

**xlOmittedCells** The cell contains a formula omitting a cell for a region.

**xlUnlockedFormulaCells** The cell which is unlocked contains a formula.

**xlEmptyCellReferences** The cell contains a formula referring to empty cells.

 [Item property as it applies to the \*\*ListColumns\*\*, \*\*ListObjects\*\*, and \*\*ListRows\*\* objects.](#)

Returns a single object from a collection.

*expression*.**Item**(*Index*)

*expression* Required. An expression that returns an object from the Applies To list.

**Index** Required **Variant**. The name or index number of the object.



[Item property as it applies to the Range object.](#)

Returns a **Range** object that represents a range at an offset to the specified range.

*expression*.**Item**(*RowIndex*, *ColumnIndex*)

*expression* Required. An expression that returns a **Range** object.

**RowIndex** Required **Variant**. The index number of the cell you want to access, in order from left to right, then down. Range.**Item**(1) returns the upper-left cell in the range; Range.**Item**(2) returns the cell immediately to the right of the upper-left cell.

**ColumnIndex** Optional **Variant**. A number or string that indicates the column number of the cell you want to access, starting with either 1 or "A" for the first column in the range.

## Remarks

Syntax 1 uses a row number and a column number or letter as index arguments. For more information about this syntax, see the [Range](#) object. The **RowIndex** and **ColumnIndex** arguments are relative offsets. In other words, specifying a **RowIndex** of 1 returns cells in the first row of the range, not the first row of the worksheet. For example, if the selection is cell C3, `Selection.Cells(2, 2)` returns cell D4 (you can use the **Item** property to index outside the original range).

[Item property as it applies to all other objects in the Applies To list.](#)

Returns a single object from a collection.

*expression*.**Item**(**Index**)

*expression* Required. An expression that returns one of the above objects.

**Index** Required **Variant**. The name or index number of the object.

## Remarks

For more information about returning a single member of a collection, see [Returning an Object from a Collection](#).

## Example

 [As it applies to the \*\*AddIns\*\* object.](#)

This example displays the status of the Analysis ToolPak add-in. Note that the string used as the index to the **AddIns** method is the **Title** property of the **AddIn** object.

```
If AddIns.Item("Analysis ToolPak").Installed = True Then
    MsgBox "Analysis ToolPak add-in is installed"
Else
    MsgBox "Analysis ToolPak add-in is not installed"
End If
```

 [As it applies to the \*\*AllowEditRanges\*\* object.](#)

This example allows edits to range ("A1:A4") on the active worksheet, notifies the user, then changes the password for this specified range and notifies the user of this change.

```
Sub UseChangePassword()

    Dim wksOne As Worksheet

    Set wksOne = Application.ActiveSheet

    ' Establish a range that can allow edits
    ' on the protected worksheet.
    wksOne.Protection.AllowEditRanges.Add _
        Title:="Classified", _
        Range:=Range("A1:A4"), _
        Password:"secret"

    MsgBox "Cells A1 to A4 can be edited on the protected worksheet."

    ' Change the password.
    wksOne.Protection.AllowEditRanges.Item(1).ChangePassword _
        Password:"moresecret"

    MsgBox "The password for these cells has been changed."

End Sub
```

 [As it applies to the \*\*Areas\*\* object.](#)

This example clears the first area in the current selection if the selection contains more than one area.

```
If Selection.Areas.Count <> 1 Then
    Selection.Areas.Item(1).Clear
End If
```

 [As it applies to the \*\*Borders\*\* object.](#)

This following example sets the color of the bottom border of cells A1:G1.

```
Worksheets("Sheet1").Range("a1:g1"). _
    Borders.Item(xlEdgeBottom).Color = RGB(255, 0, 0)
```

 [As it applies to the \*\*CalculatedMembers\*\* object.](#)

The following example notifies the user if the calculated member is valid or not. This example assumes a PivotTable exists on the active worksheet that contains either a valid or invalid calculated member.

```
Sub CheckValidity()

    Dim pvtTable As PivotTable
    Dim pvtCache As PivotCache

    Set pvtTable = ActiveSheet.PivotTables(1)
    Set pvtCache = Application.ActiveWorkbook.PivotCaches.Item(1)

    ' Handle run-time error if external source is not an OLEDB data
    On Error GoTo Not_OLEDB

    ' Check connection setting and make connection if necessary.
    If pvtCache.IsConnected = False Then
        pvtCache.MakeConnection
    End If

    ' Check if calculated member is valid.
    If pvtTable.CalculatedMembers.Item(1).IsValid = True Then
        MsgBox "The calculated member is valid."
```

```
Else
    MsgBox "The calculated member is not valid."
End If
```

```
End Sub
```

 [As it applies to the \*\*Charts\*\* object.](#)

This example sets the number of units that the trendline on Chart1 extends forward and backward. The example should be run on a 2-D column chart that contains a single series with a trendline.

```
With Charts.Item("Chart1").SeriesCollection(1).Trendlines(1)
    .Forward = 5
    .Backward = .5
End With
```

 [As it applies to the \*\*CubeFields\*\* object.](#)

This example finds the first PivotTable report whose first [cube](#) field name contains the string "Paris". The **Boolean** variable blnFoundName is set to **True** if the name is found.

```
blnFoundName = False
For Each objPT in ActiveSheet.PivotTables
    Set objCubeField = _
        objPT.CubeFields.Item(1)
    If instr(1,objCubeField.Name, "Paris") <> 0 Then
        blnFoundName = True
        Exit For
    End If
Next objPT
```

 [As it applies to the \*\*CustomProperties\*\* object.](#)

The following example demonstrates this feature. In this example, Microsoft Excel adds identifier information to the active worksheet and returns the name and value to the user.

```
Sub CheckCustomProperties()
```

```

Dim wksSheet1 As Worksheet

Set wksSheet1 = Application.ActiveSheet

' Add metadata to worksheet.
wksSheet1.CustomProperties.Add _
    Name:="Market", Value:="Nasdaq"

' Display metadata.
With wksSheet1.CustomProperties.Item(1)
    MsgBox .Name & vbTab & .Value
End With

End Sub

```

[As it applies to the \*\*Dialogs\*\* object.](#)

This example displays the **Open** dialog box and selects the **Read-Only** option.

```

Application.Dialogs.Item(xlDialogOpen).Show arg3:=True

```

[As it applies to the \*\*Filters\*\* object.](#)

The following example sets a variable to the value of the **On** property of the filter for the first column in the filtered range on the Crew worksheet.

```

Set w = Worksheets("Crew")
If w.AutoFilterMode Then
    filterIsOn = w.AutoFilter.Filters.Item(1).On
End If

```

[As it applies to the \*\*HPageBreaks\*\* object.](#)

This example changes the location of horizontal page break one.

```

Worksheets(1).HPageBreaks.Item(1).Location = .Range("e5")

```

[As it applies to the \*\*Hyperlinks\*\* object.](#)

The following example activates hyperlink one on cell E5.

```
Worksheets(1).Range("E5").Hyperlinks.Item(1).Follow
```

[As it applies to the \*\*ListObjects\*\* object.](#)

The following example displays the name of the default list object on Sheet1 of the active workbook.

```
Set wrksht = ActiveWorkbook.Worksheets("Sheet1")  
Set oListObj = wrksht.ListObjects.Item(1).Name
```

[As it applies to the \*\*Panes\*\* object.](#)

This example splits the window in which worksheet one is displayed and then scrolls through the pane in the lower-left corner of the window until row five is at the top of the pane.

```
Worksheets(1).Activate  
ActiveWindow.Split = True  
ActiveWindow.Panes.Item(3).ScrollRow = 5
```

[As it applies to the \*\*Phonetics\*\* object.](#)

This example makes the first phonetic text string in the active cell visible.

```
ActiveCell.Phonetics.Item(1).Visible = True
```

[As it applies to the \*\*PublishObjects\*\* object.](#)

This example obtains the identifier from a <DIV> tag and finds the line in a Web page (q198.htm) that you saved from a workbook. The example then creates a copy of the Web page (newq1.htm) and inserts a comment line before the <DIV> tag in the copy of the file.

```
strTargetDivID = ActiveWorkbook.PublishObjects.Item(1).DivID  
Open "\\server1\reports\q198.htm" For Input As #1  
Open "\\server1\reports\newq1.htm" For Output As #2  
While Not EOF(1)  
    Line Input #1, strFileLine
```

```
    If InStr(strFileLine, strTargetDivID) > 0 And _  
        InStr(strFileLine, "<div") > 0 Then  
        Print #2, "<!--Saved item-->"  
    End If  
    Print #2, strFileLine  
Wend  
Close #2  
Close #1
```

[As it applies to the \*\*Range\*\* object.](#)

This example fills the range A1:A10 on Sheet1, based on the contents of cell A1.

```
Worksheets("Sheet1").Range.Item("A1:A10").FillDown
```

[As it applies to the \*\*RecentFiles\*\* object.](#)

This example opens file two in the list of recently used files.

```
Application.RecentFiles.Item(2).Open
```

[As it applies to the \*\*Sheets\*\* object.](#)

This example activates Sheet1.

```
Sheets.Item("sheet1").Activate
```

[As it applies to the \*\*SmartTagRecognizer\*\* object.](#)

This example notifies the user the full name of the first smart tag recognizer.

```
MsgBox Application.SmartTagRecognizers.Item(1).FullName
```

[As it applies to the \*\*Styles\*\* object.](#)

This example changes the Normal style for the active workbook by setting the style's **Bold** property.

```
ActiveWorkbook.Styles.Item("Normal").Font.Bold = True
```

[As it applies to the \*\*VPageBreaks\*\* object.](#)

This example changes the location of vertical page break one.

```
Worksheets(1).VPageBreaks.Item(1).Location = .Range("e5")
```

[As it applies to the \*\*Windows\*\* object.](#)

This example maximizes the active window.

```
Windows.Item(1).WindowState = xlMaximized
```

[As it applies to the \*\*Workbooks\*\* object.](#)

This example sets the `wb` variable to the workbook for `Myaddin.xla`.

```
Set wb = Workbooks.Item("myaddin.xla")
```

[As it applies to the \*\*Worksheets\*\* object.](#)

**Item** is the [default member](#) for a collection. For example, the following two lines of code are equivalent.

```
ActiveWorkbook.Worksheets.Item(1)  
ActiveWorkbook.Worksheets(1)
```



# Iteration Property

**True** if Microsoft Excel will use iteration to resolve circular references.  
Read/write **Boolean**.

## Example

This example sets the **Iteration** property to **True** so that circular references will be resolved by iteration.

```
Application.Iteration = True
```



# KeepChangeHistory Property

**True** if change tracking is enabled for the shared workbook. Read/write **Boolean**.

## Example

This example sets the number of days shown in the change history for the active workbook if change tracking is enabled.

```
With ActiveWorkbook
    If .KeepChangeHistory Then
        .ChangeHistoryDuration = 7
    End If
End With
```



[Show All](#)

# KernedPairs Property

**True** if character pairs in the specified WordArt are kerned. Read/write [MsoTriState](#).

MsoTriState can be one of these MsoTriState constants.

**msoCTrue**

**msoFalse**

**msoTriStateMixed**

**msoTriStateToggle**

**msoTrue** Character pairs in the specified WordArt are kerned.

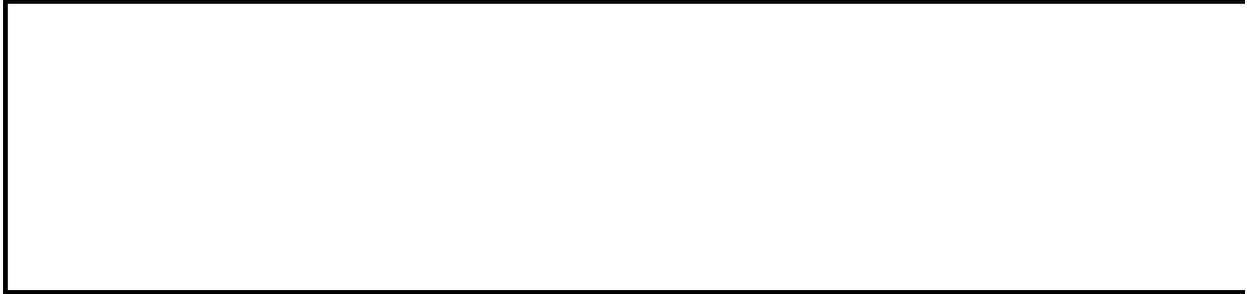
*expression*.**KernedPairs**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example turns on character pair kerning for shape three on myDocument if the shape is WordArt.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(3)
    If .Type = msoTextEffect Then
        .TextEffect.KernedPairs = msoTrue
    End If
End With
```



# KoreanCombineAux Property

When set to **True**, Microsoft Excel combines Korean auxiliary verbs and adjectives when spelling is checked. Read/write **Boolean**.

*expression*.**KoreanCombineAux**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel checks to see if the option to combine Korean auxiliary verbs and adjectives when checking spelling is on or off and notifies the user accordingly.

```
Sub KoreanSpellCheck()  
    If Application.SpellingOptions.KoreanCombineAux = True Then  
        MsgBox "The option to combine Korean auxiliary verbs and adj  
    Else  
        MsgBox "The option to combine Korean auxiliary verbs and adj  
    End If  
End Sub
```



# KoreanProcessCompound Property

When set to **True**, this enables Microsoft Excel to process Korean compound nouns when using the spelling checker. Read/write **Boolean**.

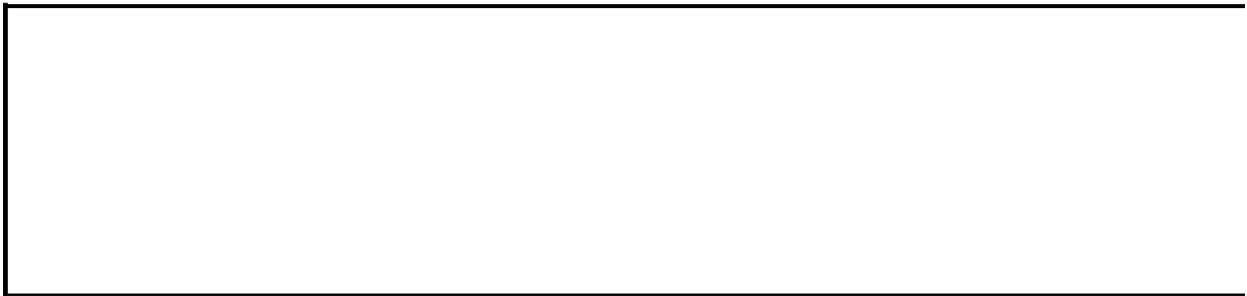
*expression*.**KoreanProcessCompound**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel checks to see if the spell checking option to process Korean compound nouns is on or off and notifies the user accordingly.

```
Sub KoreanSpellCheck()  
    If Application.SpellingOptions.KoreanProcessCompound = True Then  
        MsgBox "The spell checking feature to process Korean compoun  
    Else  
        MsgBox "The spell checking feature to process Korean compoun  
    End If  
End Sub
```



# KoreanUseAutoChangeList Property

When set to **True**, this enables Microsoft Excel to use the auto-change list for Korean words when using the spelling checker. Read/write **Boolean**.

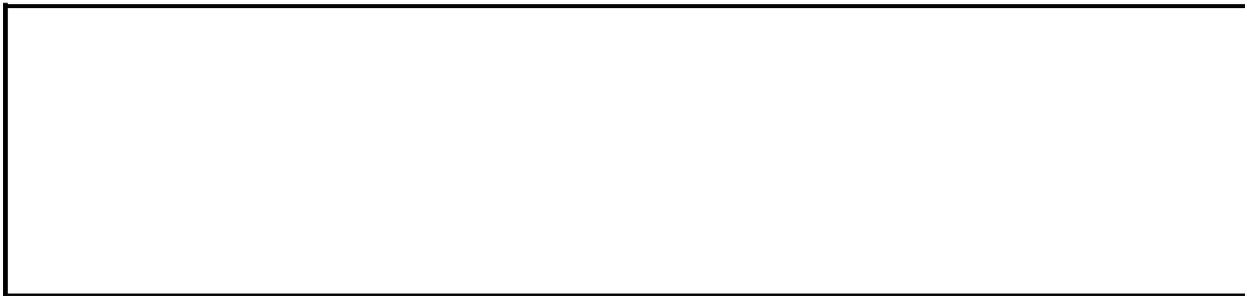
*expression*.**KoreanUseAutoChangeList**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel checks to see if the spell checking option to auto-change Korean words is on or off and notifies the user accordingly.

```
Sub KoreanSpellCheck()  
    If Application.SpellingOptions.KoreanUseAutoChangeList = True Th  
        MsgBox "The spell checking feature to auto-change Korean wor  
    Else  
        MsgBox "The spell checking feature to auto-change Korean wor  
    End If  
End Sub
```



# LabelRange Property

For a **PivotField** object, returns a [Range](#) object that represents the cell (or cells) that contain the field label. For a [PivotItem](#) object, returns a **Range** object that represents all the cells in the PivotTable report that contain the item. Read-only.

## Example

This example selects the field button for the field named "ORDER\_DATE."

```
Set pvtTable = Worksheets("Sheet1").Range("A3").PivotTable
Set pvtField = pvtTable.PivotFields("ORDER_DATE")
Worksheets("Sheet1").Activate
pvtField.LabelRange.Select
```



# LanguageSettings Property

Returns the [LanguageSettings](#) object, which contains information about the language settings in Microsoft Excel. Read-only.

*expression*.**LanguageSettings**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example returns the language identifier for the language you selected when you installed Microsoft Excel.

```
Set objLangSet = Application.LanguageSettings  
MsgBox objLangSet.LanguageID(msoLanguageIDInstall)
```



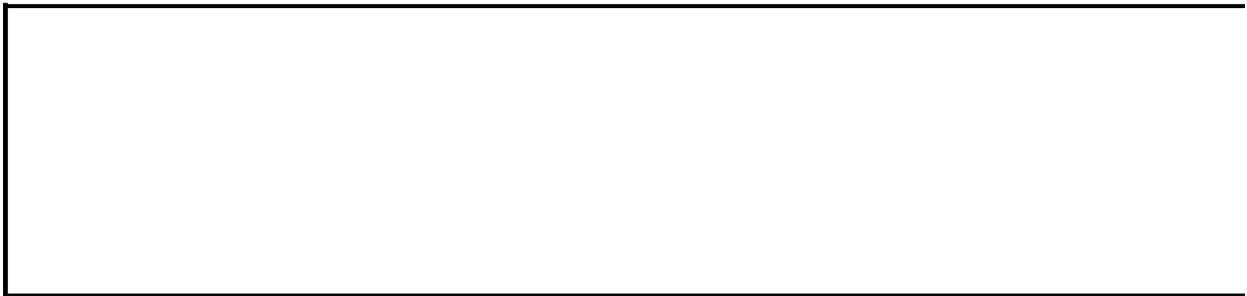
# LargeChange Property

Returns or sets the amount that the scroll box increments or decrements for a page scroll (when the user clicks in the scroll bar body region). Read/write **Long**

## Example

This example creates a scroll bar and sets its linked cell, minimum, maximum, large change, and small change values.

```
Set sb = Worksheets(1).Shapes.AddFormControl(xlScrollBar, _  
    Left:=10, Top:=10, Width:=10, Height:=200)  
With sb.ControlFormat  
    .LinkedCell = "D1"  
    .Max = 100  
    .Min = 0  
    .LargeChange = 10  
    .SmallChange = 2  
End With
```



# LastChild Property

Returns a [DiagramNode](#) object that represents the last child node of a parent node.

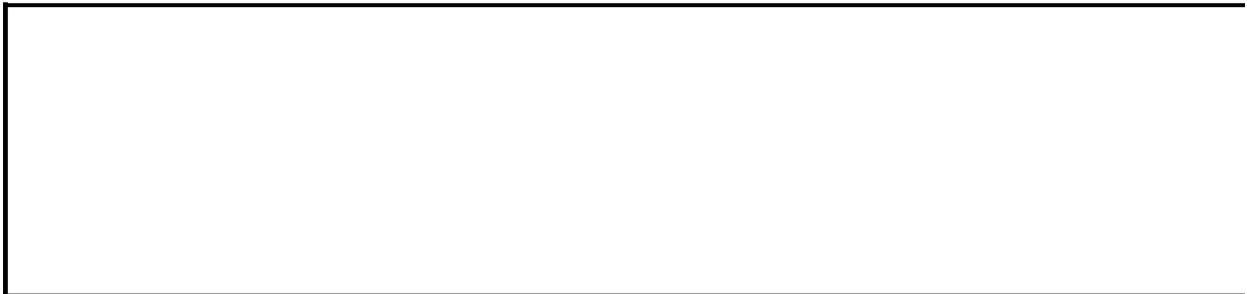
*expression*.**LastChild**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

The following example adds a child node to the last child diagram node in a newly-created diagram.

```
Sub LastChildNode()  
  
    Dim nodDiagram As DiagramNode  
    Dim nodLast As DiagramNode  
    Dim shDiagram As Shape  
    Dim intCount As Integer  
  
    Set shDiagram = ActiveSheet.Shapes.AddDiagram _  
        (Type:=msoDiagramOrgChart, Left:=10, Top:=15, _  
        Width:=400, Height:=475)  
    Set nodDiagram = shDiagram.DiagramNode.Children.AddNode  
  
    ' Add three diagram child nodes under the first diagram node  
    For intCount = 1 To 3  
        nodDiagram.Children.AddNode  
    Next intCount  
  
    ' Assign the last child node to a variable  
    Set nodLast = nodDiagram.Children.LastChild  
  
    ' Add a node to the last child node.  
    nodLast.Children.AddNode  
  
End Sub
```



[Show All](#)

# Layout Property

Returns or sets an [MsoOrgChartLayoutType](#) constant to indicate the formatting of the child nodes of an organization chart. Read/write.

MsoOrgChartLayoutType can be one of these MsoOrgChartLayoutType constants.

**msoOrgChartLayoutAssistant** Places child nodes as assistants.

**msoOrgChartLayoutBothHanging** Places child nodes vertically from the parent node on both the left and the right side.

**msoOrgChartLayoutLeftHanging** Places child nodes vertically from the parent node on the left side.

**msoOrgChartLayoutMixed** Return value for a parent node that has children formatted using more than one **MsoOrgChartLayoutType**.

**msoOrgChartLayoutRightHanging** Places child nodes vertically from the parent node on the right side.

**msoOrgChartLayoutStandard** Places child nodes horizontally below the parent node.

*expression*.Layout

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, an organization chart's layout is modified to display as right-hanging instead of standard.

```
Sub Layout()  
  
    Dim nodRoot As DiagramNode  
    Dim shDiagram As Shape  
    Dim intCount As Integer  
  
    Set shDiagram = ActiveSheet.Shapes.AddDiagram( _  
        Type:=msoDiagramOrgChart, Top:=10, _  
        Left:=15, Width:=400, Height:=475)  
  
    Set nodRoot = shDiagram.DiagramNode.Children.AddNode  
  
    ' Add three mode nodes.  
    For intCount = 1 To 3  
        nodRoot.Children.AddNode  
    Next  
  
    ' Change the layout to right-hanging.  
    nodRoot.Layout = msoOrgChartLayoutRightHanging  
  
End Sub
```



[Show All](#)

# LayoutBlankLine Property

**True** if a blank row is inserted after the specified row field in a PivotTable report. The default value is **False**. Read/write **Boolean**.

## Remarks

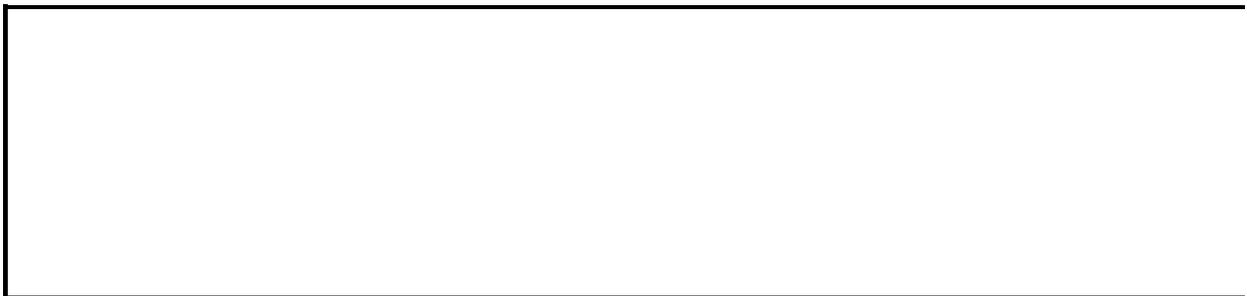
You can set this property for any PivotTable field; however, the blank row appears only if the specified field is a row field other than the innermost (lowest-level) row field. For non-[OLAP](#) data sources, the value of this property doesn't change when the field is rearranged or added to the PivotTable report.

You cannot enter data in the blank row in the PivotTable report.

## Example

This example adds a blank line after the state field in the first PivotTable report on the active worksheet.

```
With ActiveSheet.PivotTables("PivotTable1") _  
    .PivotFields("state")  
    .LayoutBlankLine = True  
End With
```



[Show All](#)

# LayoutForm Property

Returns or sets the way the specified PivotTable items appear— in table format or in outline format. Read/write [XLLayoutFormType](#).

XLLayoutFormType can be one of these XLLayoutFormType constants.

**xlTabular** Default.

**xlOutline** The [LayoutSubtotalLocation](#) property specifies where the subtotal appears in the PivotTable report.

*expression*.**LayoutForm**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

You can set this property for any PivotTable field; however, the formatting appears only if the specified field is a row field other than the innermost (lowest-level) row field. For non-[OLAP](#) data sources, the value of this property doesn't change when the field is rearranged or when it is added to or removed from the PivotTable report.

## Example

This example displays the state field in the first PivotTable report on the active worksheet in outline format, and it displays the subtotals at the top of the field.

```
With ActiveSheet.PivotTables("PivotTable1") _  
    .PivotFields("state")  
    .LayoutForm = xlOutline  
    .LayoutSubtotalLocation = xlTop  
End With
```



[Show All](#)

# LayoutPageBreak Property

**True** if a page break is inserted after each field. The default value is **False**.  
Read/write **Boolean**.

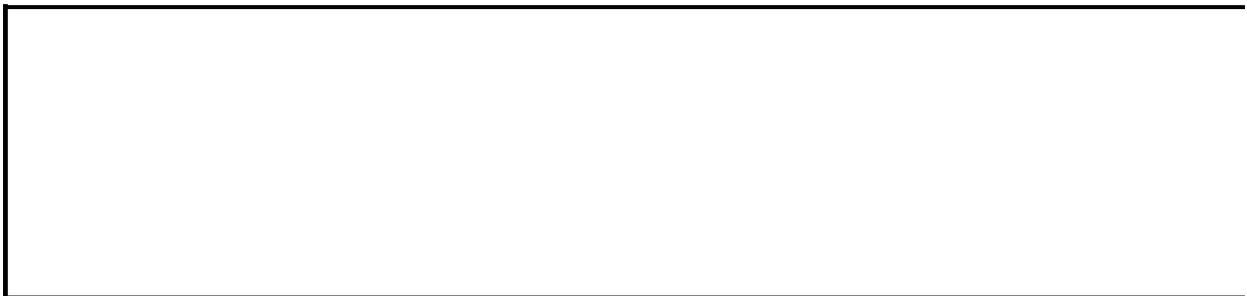
## Remarks

Although you can set this property for any PivotTable field, the print option appears only if the specified field is a row field other than the innermost (lowest-level) row field. For non-[OLAP](#) data sources, the value of this property doesn't change when the field is rearranged or when it is added to or removed from the PivotTable report.

## Example

This example adds a page break after the state field in the first PivotTable report on the active worksheet.

```
With ActiveSheet.PivotTables("PivotTable1") _  
    .PivotFields("state")  
    .LayoutPageBreak = True  
End With
```



[Show All](#)

# LayoutSubtotalLocation Property

Returns or sets the position of the PivotTable field subtotals in relation to (either above or below) the specified field. Read/write [XLSubtotalLocationType](#).

XLSubtotalLocationType can be one of these XLSubtotalLocationType constants.

**xlAtTop**

**xlAtBottom** *default*

*expression*.**LayoutSubtotalLocation**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

You can set this property for any PivotTable field in outline format; however, the formatting appears only if the specified field is a row field other than the innermost (lowest [level](#)) row field. For non- [OLAP](#) data sources, the value of this property doesn't change when the field is rearranged or when it is added to or from removed from the report.

The [LayoutForm](#) property determines whether the report appears in table format or in outline format.

## Example

This example displays the state field in the first PivotTable report on the active worksheet in outline format, and it displays the subtotals at the top of the field.

```
With ActiveSheet.PivotTables("PivotTable1") _  
    .PivotFields("state")  
    .LayoutForm = xlOutline  
    .LayoutSubtotalLocation = xlAtTop  
End With
```



[Show All](#)

# LCID Property

Returns a **Long** value that represents the LCID for the **ListColumn** object that is specified in the schema definition. In Microsoft Excel, the LCID indicates the currency symbol to be used when this is an **xListDataTypeCurrency** type. Returns 0 (which is the Language Neutral LCID) when no locale is set for the data type of the column. Read-only **Long**.

This property is used only for [lists](#) that are linked to a Microsoft Windows SharePoint Services site.

*expression*.**lcid**

*expression* Required. An expression that returns a **ListDataFormat** object.

## Remarks

In Excel, you cannot set any of the properties associated with the **ListDataFormat** object. You can set these properties, however, by modifying the list on the SharePoint site.

## Example

The following example displays the setting of the **lcid** property for the third column of the list in Sheet1 of the active workbook.

```
Sub DisplayLCID()  
    Dim wrksht As Worksheet  
    Dim objListCol As ListColumn  
  
    Set wrksht = ActiveWorkbook.Worksheets("Sheet1")  
    Set objListCol = wrksht.ListObjects(1).ListColumns(3)  
  
    MsgBox "List LCID: " & objListCol.ListDataFormat.lcid  
End Sub
```



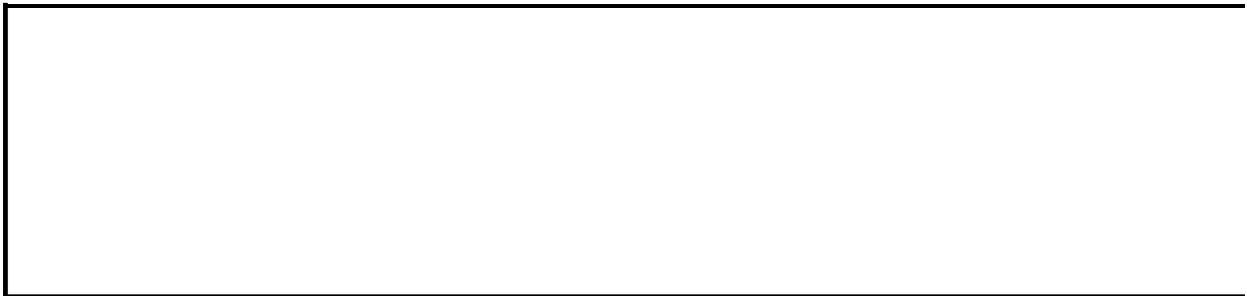
# LeaderLines Property

Returns a **LeaderLines** object that represents the leader lines for the series.  
Read-only.

## Example

This example adds data labels and blue leader lines to series one on the pie chart.

```
With Worksheets(1).ChartObjects(1).Chart.SeriesCollection(1)
    .HasDataLabels = True
    .DataLabels.Position = xlLabelPositionBestFit
    .HasLeaderLines = True
    .LeaderLines.Border.ColorIndex = 5
End With
```



[Show All](#)

# Left Property

[Left property as it applies to the \*\*Application\*\* object.](#)

The distance from the left edge of the screen to the left edge of the main Microsoft Excel window. Read/write **Double**.

*expression*.**Left**

*expression* Required. An expression that returns one of the above objects.

[Left property as it applies to the \*\*Window\*\* object.](#)

The distance from the left edge of the client area to the left edge of the window. Read/write **Double**.

*expression*.**Left**

*expression* Required. An expression that returns one of the above objects.

[Left property as it applies to the \*\*AxisTitle\*\*, \*\*ChartArea\*\*, \*\*ChartObject\*\*, \*\*ChartObjects\*\*, \*\*ChartTitle\*\*, \*\*DataLabel\*\*, \*\*DisplayUnitLabel\*\*, \*\*Legend\*\*, \*\*OLEObject\*\*, \*\*OLEObjects\*\*, and \*\*PlotArea\*\* objects.](#)

The distance from the left edge of the object to the left edge of column A (on a worksheet) or the left edge of the chart area (on a chart). Read/write **Double**.

*expression*.**Left**

*expression* Required. An expression that returns one of the above objects.

[Left property as it applies to the \*\*Axis\*\*, \*\*LegendEntry\*\*, and \*\*LegendKey\*\* objects.](#)

The distance from the left edge of the object to the left edge of the chart area. Read-only **Double**.

### *expression*.Left

*expression* Required. An expression that returns one of the above objects.

 [Left property as it applies to the \*\*Shape\*\* and \*\*ShapeRange\*\* objects.](#)

The distance from the left edge of the object to the left edge of column A (on a worksheet) or the left edge of the chart area (on a chart). Read/write **Single**.

### *expression*.Left

*expression* Required. An expression that returns one of the above objects.

 [Left property as it applies to the \*\*Range\*\* object.](#)

The distance from the left edge of column A to the left edge of the range. If the range is discontinuous, the first area is used. If the range is more than one column wide, the leftmost column in the range is used. Read-only **Variant**.

### *expression*.Left

*expression* Required. An expression that returns one of the above objects.

If the window is maximized, `Application.Left` returns a negative number that varies based on the width of the window border. Setting `Application.Left` to 0 (zero) will make the window a tiny bit smaller than it would be if the application window were maximized. In other words, if `Application.Left` is 0 (zero), the left border of the main Microsoft Excel window will just barely be visible on the screen.

## Example

This example aligns the left edge of the embedded chart with the left edge of column B.

```
With Worksheets("Sheet1")  
    .ChartObjects(1).Left = .Columns("B").Left  
End With
```



# LeftFooter Property

Returns or sets the left part of the footer. Read/write **String**.

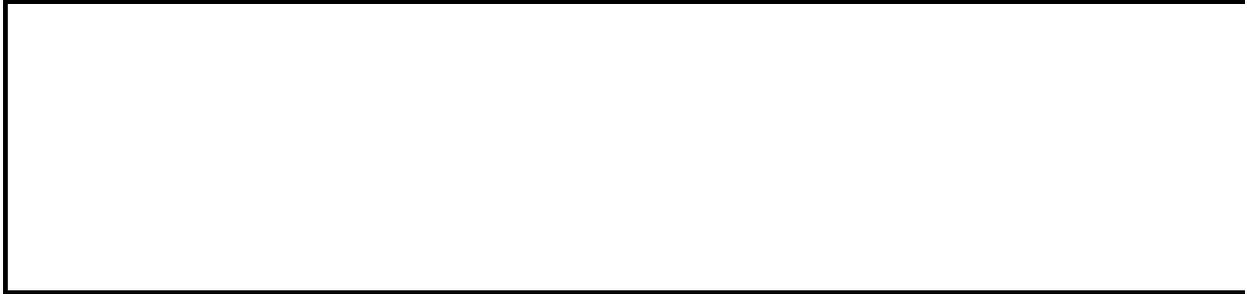
## Remarks

Special [format codes](#) can be used in the footer text.

## Example

This example prints the page number in the lower-left corner of every page.

```
Worksheets("Sheet1").PageSetup.LeftFooter = "&P"
```



# LeftFooterPicture Property

Returns a [Graphic](#) object that represents the picture for the left section of the footer. Used to set attributes about the picture.

*expression*.**LeftFooterPicture**

*expression* Required. An expression that returns a [PageSetup](#) object.

## Remarks

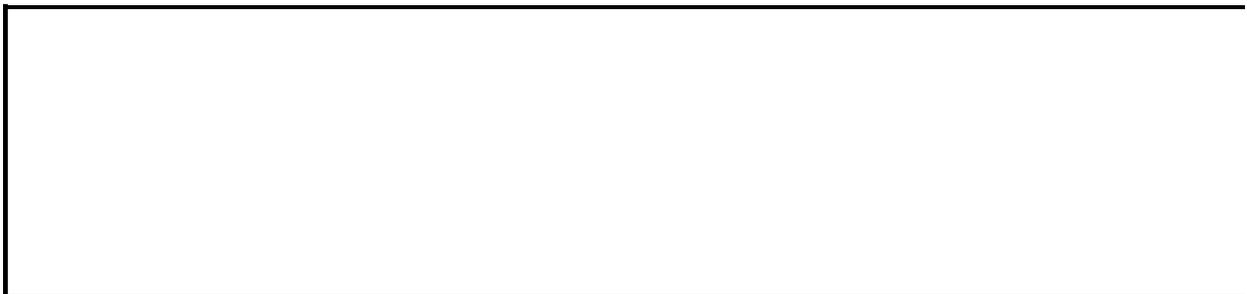
The **LeftFooterPicture** property is read-only, but the properties on it are not all read-only.

## Example

The following example adds a picture titled: Sample.jpg from the C:\ drive to the left section of the footer. This example assumes that a file called Sample.jpg exists on the C:\ drive.

```
Sub InsertPicture()  
  
    With ActiveSheet.PageSetup.LeftFooterPicture  
        .FileName = "C:\Sample.jpg"  
        .Height = 275.25  
        .Width = 463.5  
        .Brightness = 0.36  
        .ColorType = msoPictureGrayscale  
        .Contrast = 0.39  
        .CropBottom = -14.4  
        .CropLeft = -28.8  
        .CropRight = -14.4  
        .CropTop = 21.6  
    End With  
  
    ' Enable the image to show up in the left footer.  
    ActiveSheet.PageSetup.LeftFooter = "&G"  
  
End Sub
```

**Note** It is required that "&G" is a part of the **LeftFooter** property string in order for the image to show up in the left footer.



# LeftHeader Property

Returns or sets the left part of the header. Read/write **String**.

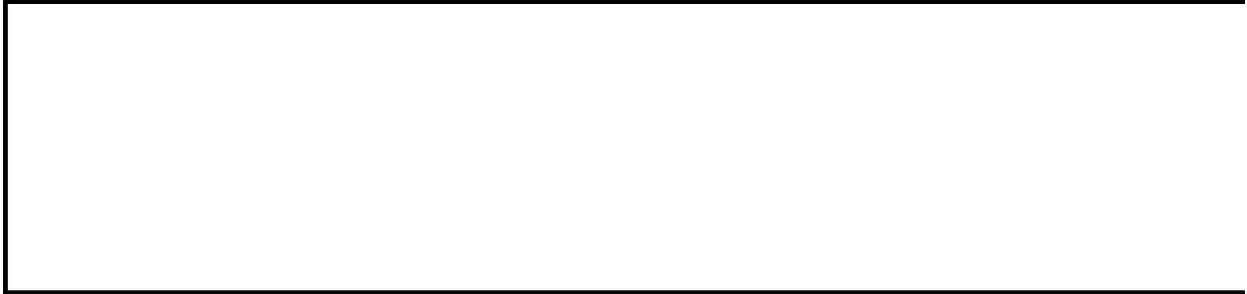
## Remarks

Special [format codes](#) can be used in the header text.

## Example

This example prints the date in the upper-left corner of every page.

```
Worksheets("Sheet1").PageSetup.LeftHeader = "&D"
```



# LeftHeaderPicture Property

Returns a [Graphic](#) object that represents the picture for the left section of the header. Used to set attributes about the picture.

*expression*.**LeftHeaderPicture**

*expression* Required. An expression that returns a [PageSetup](#) object.

## Remarks

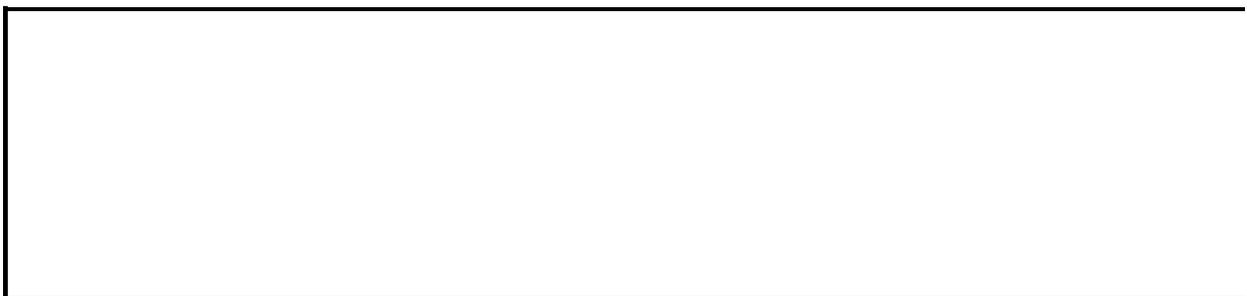
The **LeftHeaderPicture** property is read-only, but not all of its properties are read-only.

## Example

The following example adds a picture titled: Sample.jpg from the C: drive to the left section of the header. This example assumes that a file called Sample.jpg exists on the C: drive.

```
Sub InsertPicture()  
  
    With ActiveSheet.PageSetup.LeftHeaderPicture  
        .FileName = "C:\Sample.jpg"  
        .Height = 275.25  
        .Width = 463.5  
        .Brightness = 0.36  
        .ColorType = msoPictureGrayscale  
        .Contrast = 0.39  
        .CropBottom = -14.4  
        .CropLeft = -28.8  
        .CropRight = -14.4  
        .CropTop = 21.6  
    End With  
  
    ' Enable the image to show up in the left header.  
    ActiveSheet.PageSetup.LeftHeader = "&G"  
  
End Sub
```

**Note** It is required that "&G" be a part of the **LeftHeader** property string in order for the image to show up in the left header.



[Show All](#)

# LeftMargin Property

Returns or sets the size of the left margin, in [points](#). Read/write **Double**.

## Remarks

Margins are set or returned in points. Use the **InchesToPoints** method or the **CentimetersToPoints** method to convert measurements from inches or centimeters.

## Example

This example sets the left margin of Sheet1 to 1.5 inches.

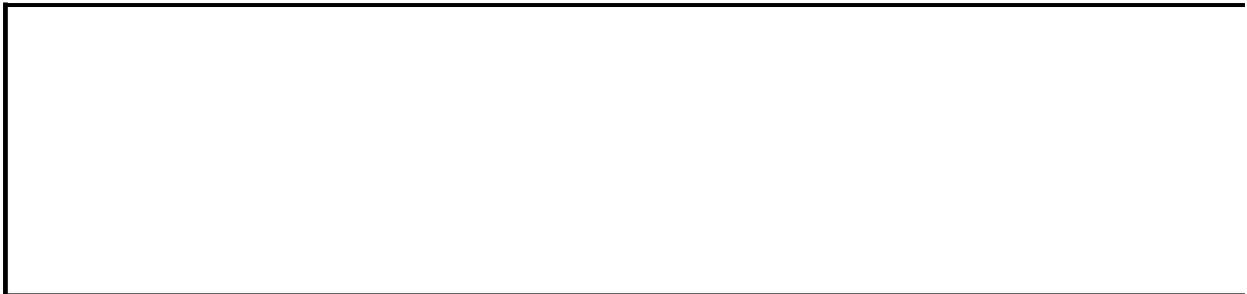
```
Worksheets("Sheet1").PageSetup.LeftMargin = _  
    Application.InchesToPoints(1.5)
```

This example sets the left margin of Sheet1 to 2 centimeters.

```
Worksheets("Sheet1").PageSetup.LeftMargin = _  
    Application.CentimetersToPoints(2)
```

This example displays the current left-margin setting for Sheet1.

```
marginInches = Worksheets("Sheet1").PageSetup.LeftMargin / _  
    Application.InchesToPoints(1)  
MsgBox "The current left margin is " & marginInches & " inches"
```



# Legend Property

Returns a [Legend](#) object that represents the legend for the chart. Read-only.

## Example

This example turns on the legend for Chart1 and then sets the legend font color to blue.

```
Charts("Chart1").HasLegend = True  
Charts("Chart1").Legend.Font.ColorIndex = 5
```



# LegendKey Property

Returns a [LegendKey](#) object that represents the legend key associated with the entry.

## Example

This example sets the legend key for legend entry one on Chart1 to be a triangle. The example should be run on a 2-D line chart.

```
Charts("Chart1").Legend.LegendEntries(1).LegendKey _  
    .MarkerStyle = xlMarkerStyleTriangle
```



[Show All](#)

# Length Property

 [Length property as it applies to the \*\*CalloutFormat\*\* object.](#)

For a **CalloutFormat** object, when the **AutoLength** property of the specified callout is set to **False**, the **Length** property returns the length (in points) of the first segment of the callout line (the segment attached to the text callout box). Applies only to callouts whose lines consist of more than one segment (types **msoCalloutThree** and **msoCalloutFour**). Read-only **Single**.

*expression.Length*

*expression* Required. An expression that returns one of the above objects.

 [Length property as it applies to the \*\*Phonetics\*\* object.](#)

For a **Phonetic** object, the **Length** property returns the number of characters of phonetic text from the position you've specified with the **Start** property. Read-only **Long**.

*expression.Length*

*expression* Required. An expression that returns one of the above objects.

## Remarks

This property is read-only. Use the [CustomLength](#) method to set the value of this property for a **CalloutFormat** object.

## Example

If the first line segment in the callout named "callout1" has a fixed length, this example specifies that the length of the first line segment in the callout named "callout2" on worksheet one will also be fixed at that length. For the example to work, both callouts must have multiple-segment lines.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes
    With .Item("callout1").Callout
        If Not .AutoLength Then len1 = .Length
    End With
    If len1 Then .Item("callout2").Callout.CustomLength len1
End With
```

This example returns the length of the second phonetic text string in the active cell.

```
ActiveCell.FormulaR1C1 = "東京都渋谷区代々木"
ActiveCell.Phonetics.Add Start:=1, Length:=3, Text:="トウキョウト"
ActiveCell.Phonetics.Add Start:=4, Length:=3, Text:="シブク"
MsgBox ActiveCell.Phonetics(2).Length
```



# LibraryPath Property

Returns the path to the Library folder, but without the final separator. Read-only **String**.

## Example

This example opens the file Oscar.xla in the Library folder.

```
pathSep = Application.PathSeparator  
f = Application.LibraryPath & pathSep & "Oscar.Xla"  
Workbooks.Open filename:=f
```



# Line Property

Returns a [LineFormat](#) object that contains line formatting properties for the specified shape. (For a line, the **LineFormat** object represents the line itself; for a shape with a border, the **LineFormat** object represents the border.) Read-only.

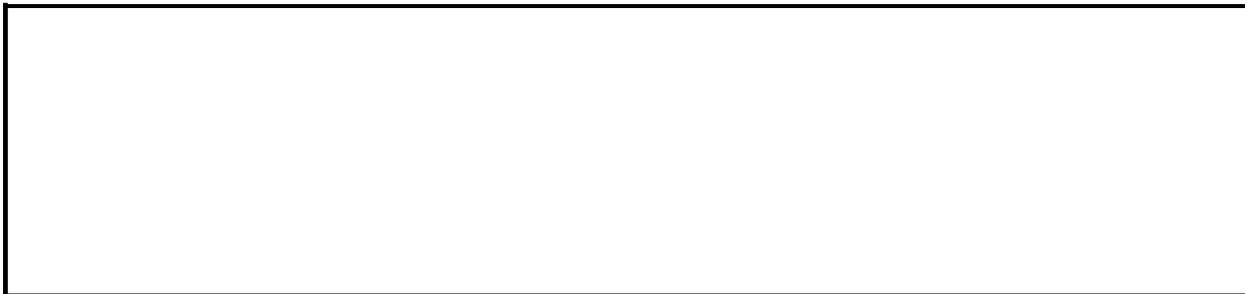
## Example

This example adds a blue dashed line to myDocument.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddLine(10, 10, 250, 250).Line
    .DashStyle = msoLineDashDotDot
    .ForeColor.RGB = RGB(50, 0, 128)
End With
```

This example adds a cross to myDocument and then sets its border to be 8 points thick and red.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddShape(msoShapeCross, 10, 10, 50, 70).Line
    .Weight = 8
    .ForeColor.RGB = RGB(255, 0, 0)
End With
```



# Line3DGroup Property

Returns a [ChartGroup](#) object that represents the line chart group on a 3-D chart.  
Read-only.

## Example

This example sets the 3-D line group in Chart1 to use a different color for each data marker.

```
Charts("Chart1").Line3DGroup.VaryByCategories = True
```



[Show All](#)

# LineStyle Property

Returns or sets the line style for the border. Read/write [XLLineStyle](#).

XLLineStyle can be one of these XLLineStyle constants.

**xlContinuous**

**xlDash**

**xlDashDot**

**xlDashDotDot**

**xlDot**

**xlDouble**

**xlSlantDashDot**

**xlLineStyleNone**

*expression*.**LineStyle**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example puts a border around the chart area and the plot area of Chart1.

```
With Charts("Chart1")  
    .ChartArea.Border.LineStyle = xlDashDot  
    With .PlotArea.Border  
        .LineStyle = xlDashDotDot  
        .Weight = xlThick  
    End With  
End With
```



# LinkedCell Property

Returns or sets the worksheet range linked to the control's value. If you place a value in the cell, the control takes this value. Likewise, if you change the value of the control, that value is also placed in the cell. Read/write **String**.

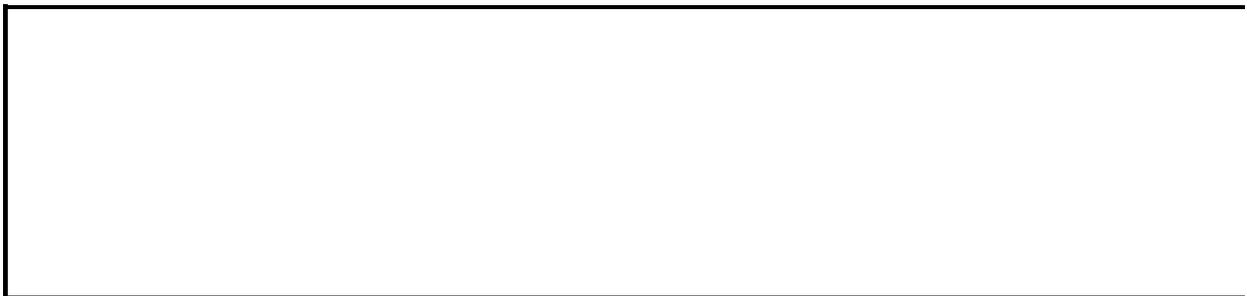
## **Remarks**

You cannot use this property with multiselect list boxes.

## Example

This example adds a check box to worksheet one and links the check box value to cell A1.

```
With Worksheets(1)
    Set cb = .Shapes.AddFormControl(xlCheckBox, 10, 10, 100, 10)
    cb.ControlFormat.LinkedCell = "A1"
End With
```



# LinkFormat Property

Returns a [LinkFormat](#) object that contains linked OLE object properties. Read-only.

## Example

This example updates all linked OLE objects on worksheet one.

```
For Each s In Worksheets(1).Shapes
    If s.Type = msoLinkedOLEObject Then s.LinkFormat.Update
Next
```



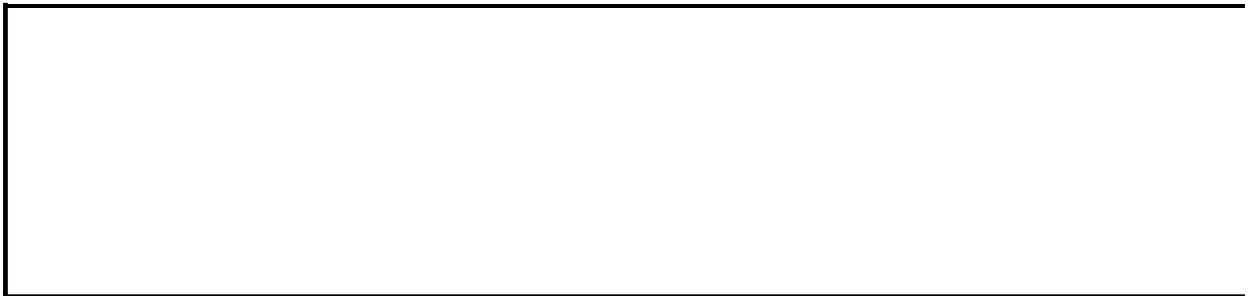
# ListChangesOnNewSheet Property

**True** if changes to the shared workbook are shown on a separate worksheet.  
Read/write **Boolean**.

## Example

This example shows changes to the shared workbook on a separate worksheet.

```
With ActiveWorkbook
    .HighlightChangesOptions _
        When:=xlSinceMyLastSave, _
        Who:="Everyone"
    .ListChangesOnNewSheet = True
End With
```



[Show All](#)

# ListColumns Property

Returns a **ListColumns** collection that represents all the columns in a **ListObject** object. Read-only.

*expression*.**ListColumns**

*expression* Required. An expression that returns a **ListObject** object.

## Example

The following example displays the name of the second column in the **ListColumns** collection object as created by a call to the **ListColumns** property. For this code to run, the Sheet1 worksheet must contain a [list](#).

```
Sub DisplayColumnName
    Dim wrksht As Worksheet
    Dim objListObj As ListObject
    Dim objListCols As ListColumns

    Set wrksht = ActiveWorkbook.Worksheets("Sheet1")
    Set objListObj = wrksht.ListObjects(1)
    Set objListCols = objListObj.ListColumns

    Debug.Print objListCols(2).Name
End Sub
```



# ListCount Property

Returns the number of entries in a list box or combo box. Returns 0 (zero) if there are no entries in the list. Read-only **Long**.

## Example

This example adjusts a combo box to display all entries in its list. If `Shapes(1)` does not represent a combo box, this example fails.

```
Set cf = Worksheets(1).Shapes(1).ControlFormat  
cf.DropDownLines = cf.ListCount
```



# ListDataFormat Property

Returns a **ListDataFormat** object for the **ListColumn** object. Read-only.

*expression*.**ListDataFormat**

*expression* Required. An expression that returns a **ListColumn** object.

## Remarks

Use the **ListDataFormat** property to return a **ListDataFormat** object.

The following code returns a **ListDataFormat** object for the Products column of a list in the active worksheet.

```
Dim objListDataFormat as ListDataFormat  
set objListDataFormat = ActiveSheet.ListColumns("Products").ListData
```



# ListDataValidation Property

A **Boolean** value that is **True** if data validation is enabled in a list. Read/write **Boolean**.

*expression*.**ListDataValidation**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Data validation for a list can also be turned on or off by in the user interface by selecting or clearing the **List data validation error** check box on the **Error Checking** tab of the **Options** dialog box.

## Example

```
Sub PrintListValidationOnOrOff()  
    Debug.Print Application.ErrorCheckingOptions.ListDataValidation  
End Sub
```



# ListFillRange Property

Returns or sets the worksheet range used to fill the specified list box. Setting this property destroys any existing list in the list box. Read/write **String**.

## Remarks

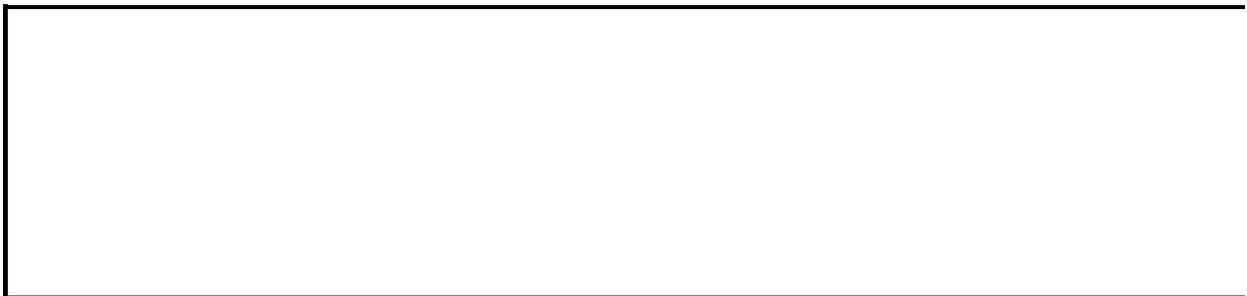
Microsoft Excel reads the contents of every cell in the range and inserts the cell values into the list box. The list tracks changes in the range's cells.

If the list in the list box was created with the **AddItem** method, this property returns an empty string ("").

## Example

This example adds a list box to worksheet one and sets the fill range for the list box.

```
With Worksheets(1)
    Set lb = .Shapes.AddFormControl(xlListBox, 100, 10, 100, 100)
    lb.ControlFormat.ListFillRange = "A1:A10"
End With
```



# ListHeaderRows Property

Returns the number of header rows for the specified range. Read-only **Long**.

## Remarks

Before you use this property, use the [CurrentRegion](#) property to find the boundaries of the range.

## Example

This example sets the `rTbl` variable to the range represented by the current region for the active cell, not including any header rows.

```
Set rTbl = ActiveCell.CurrentRegion
' remove the headers from the range
iHdrRows = rTbl.ListHeaderRows
If iHdrRows > 0 Then
    ' resize the range minus n rows
    Set rTbl = rTbl.Resize(rTbl.Rows.Count - iHdrRows)
    ' and then move the resized range down to
    ' get to the first non-header row
    Set rTbl = rTbl.Offset(iHdrRows)
End If
```



# ListIndex Property

Returns or sets the index number of the currently selected item in a list box or combo box. Read/write **Long**.

## **Remarks**

You cannot use this property with multiselect list boxes.

## Example

This example removes the selected item from a list box. If Shapes(2) doesn't represent a list box, this example fails.

```
Set lbcf = Worksheets(1).Shapes(2).ControlFormat  
lbcf.RemoveItem lbcf.ListIndex
```



# ListObject Property

Returns a **ListObject** object for the [Range](#) object or [QueryTable](#) object. Read-only **ListObject** object.

*expression*.**ListObject**

*expression* Required. An expression that returns one of the objects in the Applies To list.



# ListObjects Property

Returns a collection of **ListObject** objects in the worksheet. Read-only **ListObjects** collection.

*expression*.**ListObjects**

*expression* Required. An expression that returns one of the objects in the Applies To list.



# ListRows Property

Returns a **ListRows** object that represents all the rows of data in the **ListObject** object. Read-only.

*expression*.**ListRows**

*expression* Required. An expression that returns a **ListObject** object.

## Remarks

The **ListRows** object returned does not include the header, total, or Insert rows.

## Example

The following example deletes a row specified by number in the **ListRows** collection that is created by a call to the **ListRows** property.

```
Sub DeleteListRow(iRowNumber As Integer)
    Dim wrksht As Worksheet
    Dim objListObj As ListObject
    Dim objListRows As ListRows

    Set wrksht = ActiveWorkbook.Worksheets("Sheet1")
    Set objListObj = wrksht.ListObjects(1)
    Set objListRows = objListObj.ListRows

    If (iRowNumber <> 0) And (iRowNumber < objListRows.Count - 1) Then
        objListRows(iRowNumber).Delete
    End If
End Sub
```



# ListSelection Property

Sets or returns a **Long** value that represents the index number of the selected item in a smart document list box control. Read/write.

*expression*.**ListSelection**

*expression* Required. An expression that returns a [SmartTagAction](#) object.

## Remarks

For more information on smart documents, please see the Smart Document Software Development Kit on the [Microsoft Developer Network \(MSDN\)](#) Web site.

---

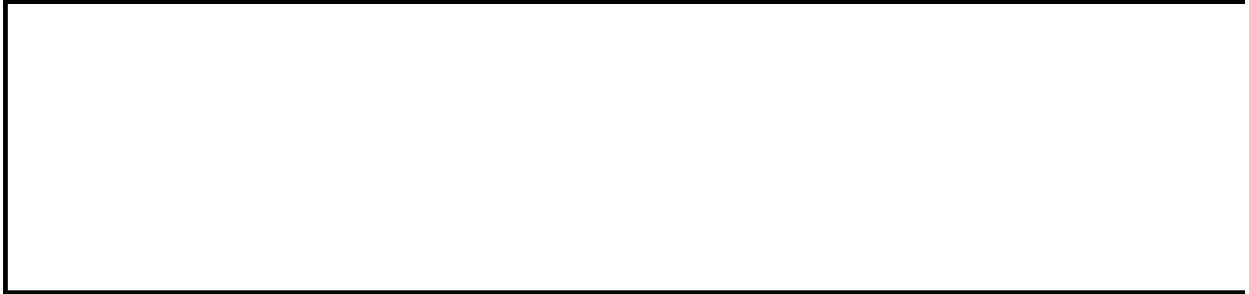
# LoadPictures Property

**True** if images are loaded when you open a document in Microsoft Excel, usually when the images and document were not created in Microsoft Excel. **False** if the images are not loaded. The default value is **True**. Read/write **Boolean**.

## Example

This example causes images to load when the document is opened in Excel.

```
Application.DefaultWebOptions.LoadPictures = True
```



[Show All](#)

# LocalConnection Property

Returns or sets the connection string to an [offline cube file](#). Read/write **String**.

## Remarks

For a non-[OLAP](#) data source, the value of the **LocalConnection** property is an empty string, and the [UseLocalConnection](#) property is set to **False**.

Setting the **LocalConnection** property doesn't immediately initiate the connection to the data source. You must first use the [Refresh](#) method to make the connection and retrieve the data.

The value of the **LocalConnection** property is used if the **UseLocalConnection** property is set to **True**. If the **UseLocalConnection** property is set to **False**, the [Connection](#) property specifies the connection string for query tables based on sources other than local cube files.

For more information about the syntax for connection strings, see the Help topic for the [Add](#) method of the **PivotTables** collection.

## Example

This example sets the connection string of the first PivotTable cache to reference an [offline cube file](#).

```
With ActiveWorkbook.PivotCaches(1)
    .LocalConnection = _
        "OLEDB;Provider=MSOLAP;Data Source=C:\Data\DataCube.cub"
    .UseLocalConnection = True
End With
```



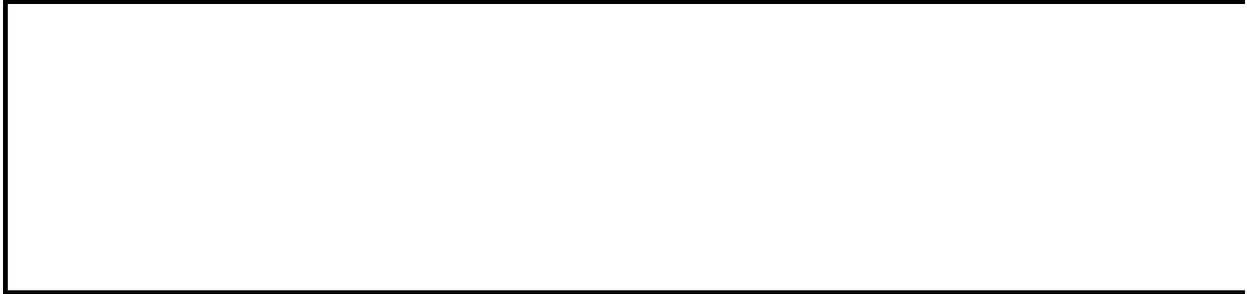
# Location Property

For the **HPageBreak** and **VPageBreak** objects, this property returns or sets the cell (a **Range** object) that defines the page-break location. Horizontal page breaks are aligned with the top edge of the location cell; vertical page breaks are aligned with the left edge of the location cell. Read/write **Range**.

## Example

This example moves the horizontal page-break location.

```
Worksheets(1).HPageBreaks(1).Location = Worksheets(1).Range("e5")
```



[Show All](#)

# LocationInTable Property

Returns a constant that describes the part of the [PivotTable](#) report that contains the upper-left corner of the specified range. Can be one of the following [XLLocationInTable](#). constants. Read-only **Long**.

XLLocationInTable can be one of these XLLocationInTable constants.

**xlRowHeader**

**xlColumnHeader**

**xlPageHeader**

**xlDataHeader**

**xlRowItem**

**xlColumnItem**

**xlPageItem**

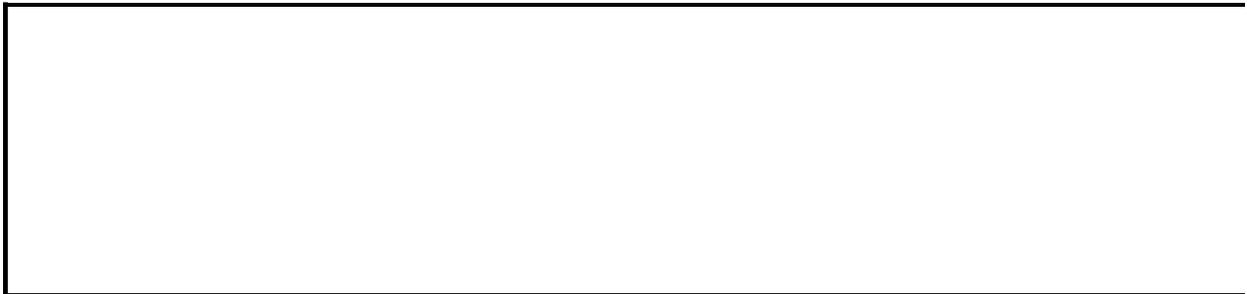
**xlDataItem**

**xlTableBody**

## Example

This example displays a message box that describes the location of the active cell within the PivotTable report.

```
Worksheets("Sheet1").Activate
Select Case ActiveCell.LocationInTable
Case Is = xlRowHeader
    MsgBox "Active cell is part of a row header"
Case Is = xlColumnHeader
    MsgBox "Active cell is part of a column header"
Case Is = xlPageHeader
    MsgBox "Active cell is part of a page header"
Case Is = xlDataHeader
    MsgBox "Active cell is part of a data header"
Case Is = xlRowItem
    MsgBox "Active cell is part of a row item"
Case Is = xlColumnItem
    MsgBox "Active cell is part of a column item"
Case Is = xlPageItem
    MsgBox "Active cell is part of a page item"
Case Is = xlDataItem
    MsgBox "Active cell is part of a data item"
Case Is = xlTableBody
    MsgBox "Active cell is part of the table body"
End Select
```



# LocationOfComponents Property

Returns or sets the central URL (on the intranet or Web) or path (local or network) to the location from which authorized users can download Microsoft Office Web components when viewing your saved document. The default value is the local or network installation path for Microsoft Office. Read/write **String**.

## Remarks

Office Web components are automatically downloaded with the specified Web page if the [DownloadComponents](#) property is set to **True**, the components are not already installed, the path is valid and points to a location that contains the necessary components, and the user has a valid Microsoft Office 2000 license.

## Example

This example sets the path to the location from which users can download Microsoft Office Web components.

```
Application.DefaultWebOptions.DownloadComponents = True  
Application.DefaultWebOptions.LocationOfComponents = _  
    Application.Path & Application.PathSeparator & "foo"
```



[Show All](#)

# LockAspectRatio Property

**True** if the specified shape retains its original proportions when you resize it.  
**False** if you can change the height and width of the shape independently of one another when you resize it. Read/write [MsoTriState](#).

MsoTriState can be one of these MsoTriState constants.

**msoCTrue**

**msoFalse** You can change the height and width of the shape independently of one another when you resize it.

**msoTriStateMixed**

**msoTriStateToggle**

**msoTrue** The specified shape retains its original proportions when you resize it.

*expression*.**LockAspectRatio**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example adds a cube to myDocument. The cube can be moved and resized, but not reproportioned.

```
Set myDocument = Worksheets(1)
myDocument.Shapes.AddShape(msoShapeCube, _
    50, 50, 100, 200).LockAspectRatio = msoTrue
```



[Show All](#)

# Locked Property

[Locked property as it applies to the \*\*ChartObject\*\*, \*\*ChartObjects\*\*, \*\*LinkFormat\*\*, \*\*OLEObject\*\*, \*\*OLEObjects\*\*, \*\*Scenario\*\*, \*\*Shape\*\*, and \*\*Style\*\* objects.](#)

**True** if the object is locked, **False** if the object can be modified when the sheet is protected. Read/write **Boolean**.

*expression*.**Locked**

*expression* Required. An expression that returns one of the above objects.

[Locked property as it applies to the \*\*CellFormat\*\* and \*\*Range\*\* objects.](#)

**True** if the object is locked, **False** if the object can be modified when the sheet is protected. Returns **Null** if the specified range contains both locked and unlocked cells. Read/write **Variant**.

*expression*.**Locked**

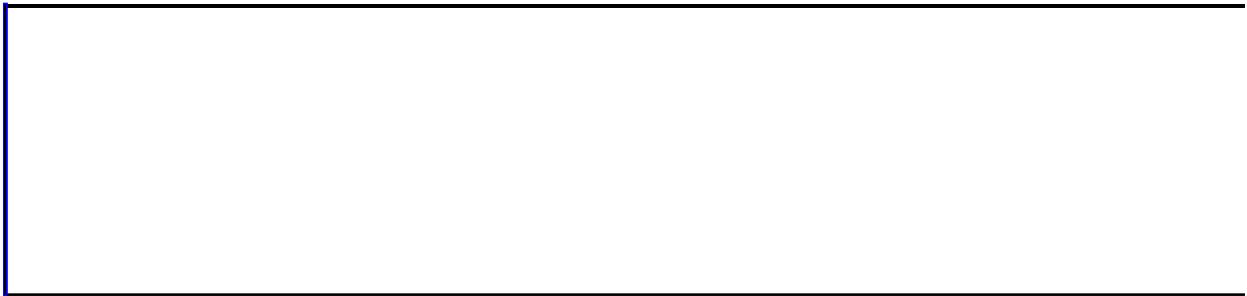
*expression* Required. An expression that returns one of the above objects.

## Example

[As it applies to the \*\*CellFormat\*\* and \*\*Range\*\* objects.](#)

This example unlocks cells A1:G37 on Sheet1 so that they can be modified when the sheet is protected.

```
Worksheets("Sheet1").Range("A1:G37").Locked = False  
Worksheets("Sheet1").Protect
```



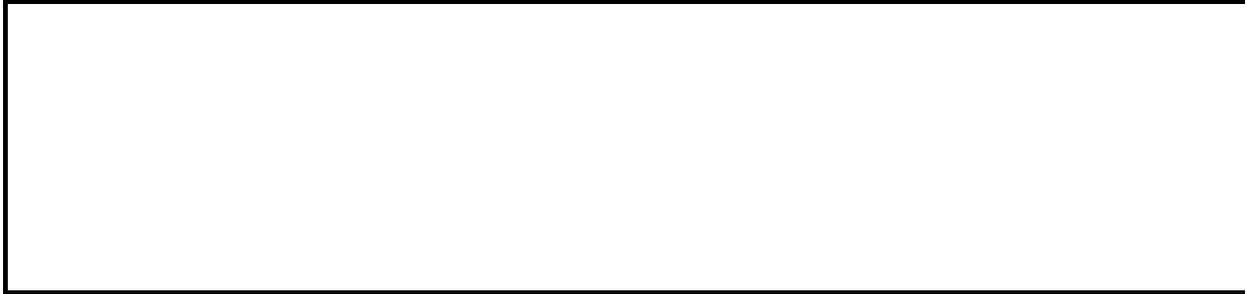
# LockedText Property

**True** if the text in the specified object will be locked to prevent changes when the workbook is protected. Read/write **Boolean**.

## Example

This example locks text in embedded chart one when the workbook is protected.

```
Worksheets(1).ChartObjects(1).LockedText = True
```



[Show All](#)

# MacroType Property

Returns or sets what the name refers to. Read/write [xlXLMMacroType](#).

xlXLMMacroType can be one of these xlXLMMacroType constants.

**xlCommand.** The name refers to a user-defined macro.

**xlFunction.** The name refers to a user-defined function.

**xlNotXLM.** The name doesn't refer to a function or macro.

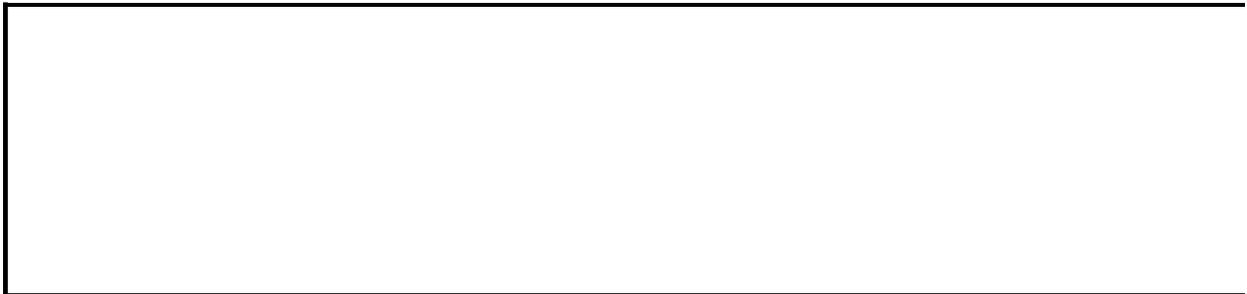
*expression*.**MacroType**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example assumes that you created a custom function or command on a Microsoft Excel version 4.0 macro sheet. The example displays the function category, in the language of the macro. It assumes that the name of the custom function or command is the only name in the workbook.

```
With ActiveWorkbook.Names(1)
  If .MacroType <> xlNotXLM Then
    MsgBox "The category for this name is " & .Category
  Else
    MsgBox "This name does not refer to" & _
      " a custom function or command."
  End If
End With
```



# MailEnvelope Property

Represents an e-mail header for a document.

*expression*.**MailEnvelope**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example sets the comments for the header of the active worksheet.

```
Sub HeaderComments()
```

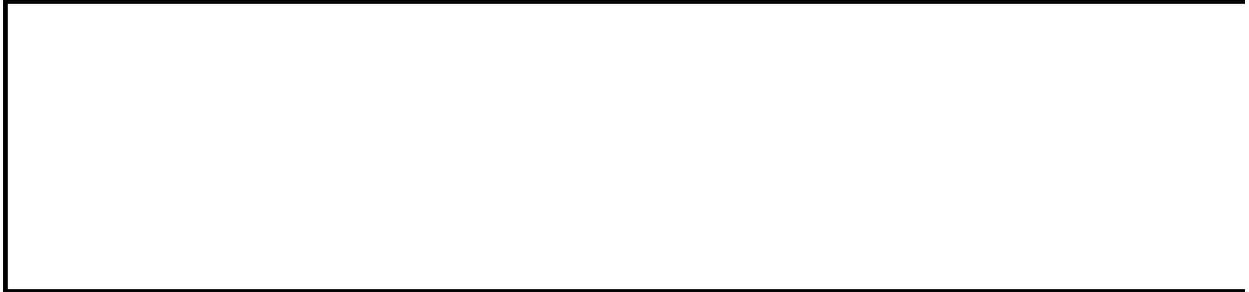
```
    ActiveSheet.MailEnvelope.Introduction = "To Whom It May Concern:"
```

```
End Sub
```



# Mailer Property

You have requested Help for a Visual Basic keyword used only on the Macintosh. For information about this keyword, consult the language reference Help included with Microsoft Office Macintosh Edition.



# MailSession Property

Returns the MAPI mail session number as a hexadecimal string (if there's an active session), or returns **Null** if there's no session. Read-only **Variant**.

## **Remarks**

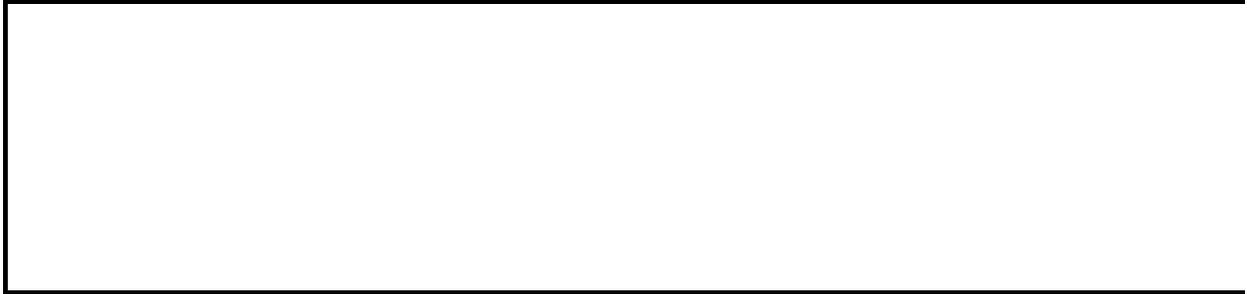
This property applies only to mail sessions created by Microsoft Excel (it doesn't return a mail session number for Microsoft Mail).

This property isn't used on PowerTalk mail systems.

## Example

This example closes the established mail session, if there is one.

```
If Not IsNull(Application.MailSession) Then Application.MailLogoff
```



[Show All](#)

# MailSystem Property

Returns the mail system that's installed on the host machine. Read-only [XIMailSystem](#).

XIMailSystem can be one of these XIMailSystem constants.

**xlMAPI**

**xlNoMailSystem**

**xlPowerTalk**

*expression*.**MailSystem**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example displays the name of the mail system that's installed on the computer.

```
Select Case Application.MailSystem
  Case xlMAPI
    MsgBox "Mail system is Microsoft Mail"
  Case xlPowerTalk
    MsgBox "Mail system is PowerTalk"
  Case xlNoMailSystem
    MsgBox "No mail system installed"
End Select
```



[Show All](#)

# MaintainConnection Property

**True** if the connection to the specified data source is maintained after the refresh and until the workbook is closed. The default value is **True**. Read/write **Boolean**.

## Remarks

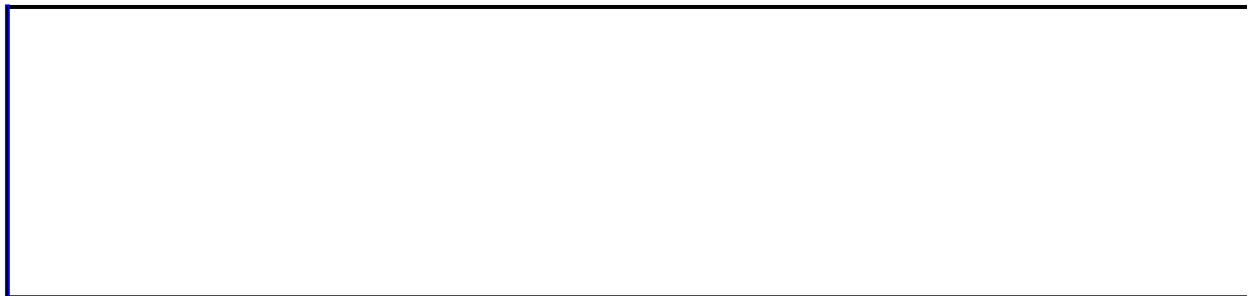
You can set the **MaintainConnection** property only if the [QueryType](#) property of the query table or PivotTable cache is set to **xlOLEDBQuery**.

If you anticipate frequent queries to a server, setting this property to **True** might improve performance by reducing reconnection time. Setting the property to **False** causes an open connection to be closed.

## Example

This example creates a new PivotTable cache based on an [OLAP provider](#), and then it creates a new PivotTable report based on the cache, at cell A3 on the active worksheet. The example terminates the connection after the initial refresh.

```
With ActiveWorkbook.PivotCaches.Add(SourceType:=xlExternal)
    .Connection = _
        "OLEDB;Provider=MSOLAP;Location=srvdata;Initial Catalog=Nati
    .MaintainConnection = False
    .CreatePivotTable TableDestination:=Range("A3"), _
        TableName:= "PivotTable1"
End With
With ActiveSheet.PivotTables("PivotTable1")
    .SmallGrid = False
    .PivotCache.RefreshPeriod = 0
    With .CubeFields("[state]")
        .Orientation = xlColumnField
        .Position = 0
    End With
    With .CubeFields("[Measures].[Count Of au_id]")
        .Orientation = xlDataField
        .Position = 0
    End With
End With
```



# MajorGridlines Property

Returns a [Gridlines](#) object that represents the major gridlines for the specified axis. Only axes in the primary axis group can have gridlines. Read-only.

## Example

This example sets the color of the major gridlines for the value axis in Chart1.

```
With Charts("Chart1").Axes(xlValue)
    If .HasMajorGridlines Then
        .MajorGridlines.Border.ColorIndex = 5    'set color to blue
    End If
End With
```



[Show All](#)

# MajorTickMark Property

Returns or sets the type of major tick mark for the specified axis. Read/write [xlTickMark](#).

XlTickMark can be one of these XlTickMark constants.

**xlTickMarkInside**

**xlTickMarkOutside**

**xlTickMarkCross**

**xlTickMarkNone**

*expression*.**MajorTickMark**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example sets the major tick marks for the value axis in Chart1 to be outside the axis.

```
Charts("Chart1").Axes(xlValue).MajorTickMark = xlTickMarkOutside
```



# MajorUnit Property

Returns or sets the major units for the axis. Read/write **Double**.

## Remarks

Setting this property sets the [MajorUnitIsAuto](#) property to **False**.

Use the [TickMarkSpacing](#) property to set tick mark spacing on the category axis.

## Example

This example sets the major and minor units for the value axis in Chart1.

```
With Charts("Chart1").Axes(xlValue)  
    .MajorUnit = 100  
    .MinorUnit = 20  
End With
```



# MajorUnitIsAuto Property

**True** if Microsoft Excel calculates the major units for the axis. Read/write **Boolean**.

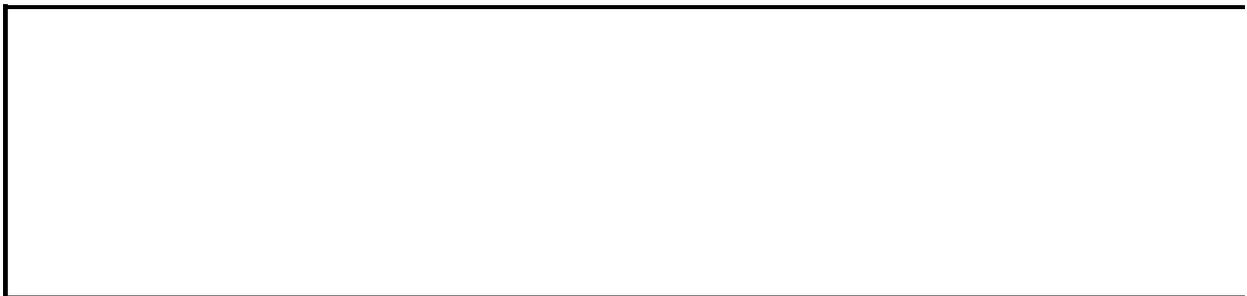
## Remarks

Setting the [MajorUnit](#) property sets this property to **False**.

## Example

This example automatically sets the major and minor units for the value axis in Chart1.

```
With Charts("Chart1").Axes(xlValue)  
    .MajorUnitIsAuto = True  
    .MinorUnitIsAuto = True  
End With
```



[Show All](#)

# MajorUnitScale Property

Returns or sets the major unit scale value for the category axis when the **CategoryType** property is set to **xlTimeScale**. Read/write [XLTimeUnit](#).

XLTimeUnit can be one of these XLTimeUnit constants.

**xlMonths**

**xlDays**

**xlYears**

*expression*.**MajorUnitScale**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example sets the category axis to use a time scale and sets the major and minor units.

```
With Charts(1).Axes(xlCategory)
    .CategoryType = xlTimeScale
    .MajorUnit = 5
    .MajorUnitScale = xlDays
    .MinorUnit = 1
    .MinorUnitScale = xlDays
End With
```



# ManualUpdate Property

**True** if the PivotTable report is recalculated only at the user's request. The default value is **False**. Read/write **Boolean**.

*expression*.**ManualUpdate**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

This property is set to **False** immediately after your program terminates and after you execute the statement in the Immediate window of the Microsoft Visual Basic Editor.

## Example

This example causes the PivotTable report to be recalculated only at the user's request.

```
Worksheets(1).PivotTables("Pivot1").ManualUpdate = True
```



# Map Property

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Returns an [XmlMap](#) object that represents the schema map that contains the specified [XPath](#) object. Read-only.

*expression*.**Map**

*expression* Required. An expression that returns an **XPath** object.



# MapPaperSize Property

**True** if documents formatted for the standard paper size of another country/region (for example, A4) are automatically adjusted so that they're printed correctly on the standard paper size (for example, Letter) of your country/region. Read/write **Boolean**.

*expression*.**MapPaperSize**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example determines if Microsoft Excel can adjust the paper size according to the country/region setting and notifies the user.

```
Sub UseMapPaperSize()  
    ' Determine setting and notify user.  
    If Application.MapPaperSize = True Then  
        MsgBox "Microsoft Excel automatically " & _  
            "adjusts the paper size according to the country/region  
    Else  
        MsgBox "Microsoft Excel does not " & _  
            "automatically adjusts the paper size according to the c  
    End If  
End Sub
```



# MarginBottom Property

Returns or sets the distance (in points) between the bottom of the text frame and the bottom of the inscribed rectangle of the shape that contains the text.

Read/write **Single**.

## Example

This example adds a rectangle to myDocument, adds text to the rectangle, and then sets the margins for the text frame.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddShape(msoShapeRectangle, _
    0, 0, 250, 140).TextFrame
    .AutoMargins = False
    .Characters.Text = "Here is some test text"
    .MarginBottom = 0
    .MarginLeft = 100
    .MarginRight = 0
    .MarginTop = 20
End With
```



# MarginLeft Property

Returns or sets the distance (in points) between the left edge of the text frame and the left edge of the inscribed rectangle of the shape that contains the text. Read/write **Single**.

## Example

This example adds a rectangle to myDocument, adds text to the rectangle, and then sets the margins for the text frame.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddShape(msoShapeRectangle, _
    0, 0, 250, 140).TextFrame
    .AutoMargins = False
    .Characters.Text = "Here is some test text"
    .MarginBottom = 0
    .MarginLeft = 100
    .MarginRight = 0
    .MarginTop = 20
End With
```



# MarginRight Property

Returns or sets the distance (in points) between the right edge of the text frame and the right edge of the inscribed rectangle of the shape that contains the text. Read/write **Single**.

## Example

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddShape(msoShapeRectangle, _
    0, 0, 250, 140).TextFrame
    .AutoMargins = False
    .Characters.Text = "Here is some test text"
    .MarginBottom = 0
    .MarginLeft = 100
    .MarginRight = 0
    .MarginTop = 20
End With
```



# MarginTop Property

Returns or sets the distance (in points) between the top of the text frame and the top of the inscribed rectangle of the shape that contains the text. Read/write **Single**.

## Example

This example adds a rectangle to myDocument, adds text to the rectangle, and then sets the margins for the text frame.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddShape(msoShapeRectangle, _
    0, 0, 250, 140).TextFrame
    .AutoMargins = False
    .Characters.Text = "Here is some test text"
    .MarginBottom = 0
    .MarginLeft = 100
    .MarginRight = 0
    .MarginTop = 20
End With
```



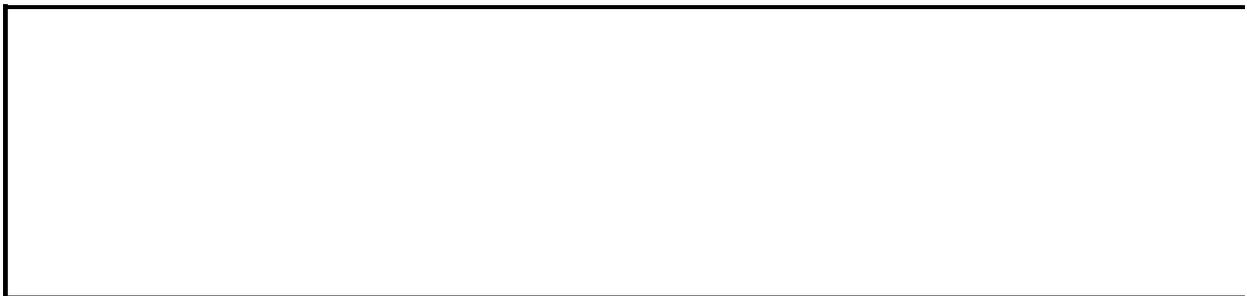
# MarkerBackgroundColor Property

Returns or sets the marker background color as an RGB value. Applies only to line, scatter, and radar charts. Read/write **Long**.

## Example

This example sets the marker background and foreground colors for the second point in series one in Chart1.

```
With Charts("Chart1").SeriesCollection(1).Points(2)  
    .MarkerBackgroundColor = RGB(0,255,0)    ' green  
    .MarkerForegroundColor = RGB(255,0,0)    ' red  
End With
```

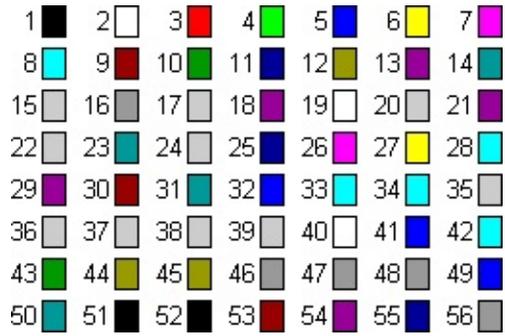


# MarkerBackgroundColorIndex Property

Returns or sets the marker background color as an index into the current color palette, or as one of the following **XlColorIndex** constants: **xlColorIndexAutomatic** or **xlColorIndexNone**. Applies only to line, scatter, and radar charts. Read/write **Long**.

# Remarks

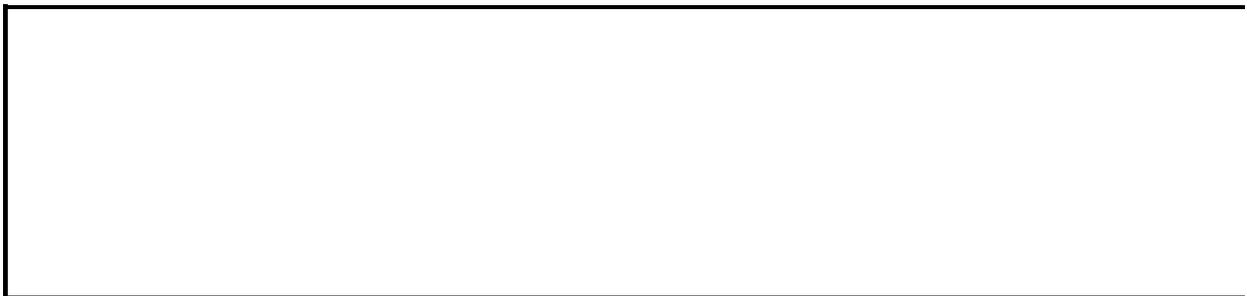
The following illustration shows the color-index values in the default color palette.



## Example

This example sets the marker background and foreground colors for the second point in series one in Chart1.

```
With Charts("Chart1").SeriesCollection(1).Points(2)  
    .MarkerBackgroundColorIndex = 4    'green  
    .MarkerForegroundColorIndex = 3    'red  
End With
```



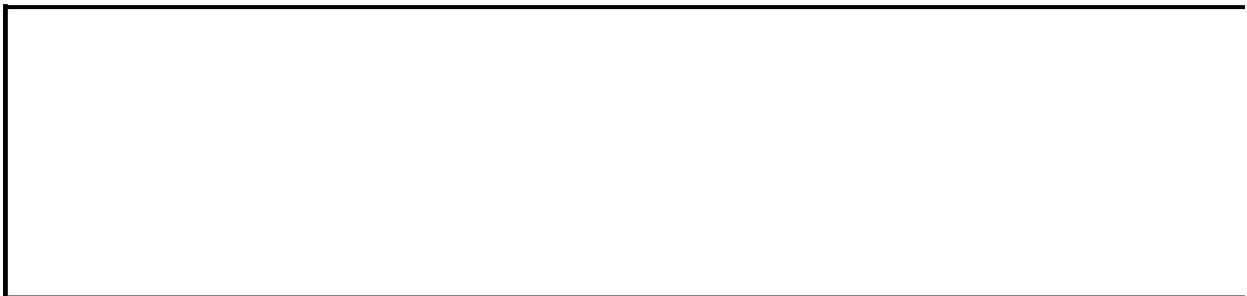
# MarkerForegroundColor Property

Returns or sets the foreground color of the marker as an RGB value. Applies only to line, scatter, and radar charts. Read/write **Long**.

## Example

This example sets the marker background and foreground colors for the second point in series one in Chart1.

```
With Charts("Chart1").SeriesCollection(1).Points(2)  
    .MarkerBackgroundColor = RGB(0,255,0)    ' green  
    .MarkerForegroundColor = RGB(255,0,0)    ' red  
End With
```

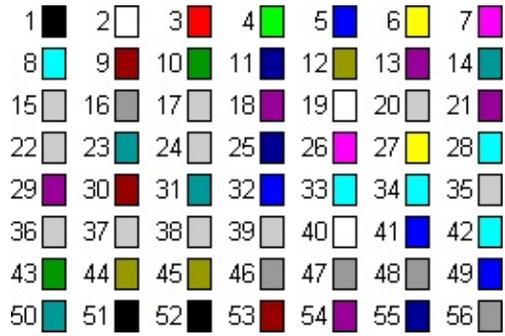


# MarkerForegroundColorIndex Property

Returns or sets the marker foreground color as an index into the current color palette, or as one of the following **XIColorIndex** constants: **xIColorIndexAutomatic** or **xIColorIndexNone**. Applies only to line, scatter, and radar charts. Read/write **Long**.

# Remarks

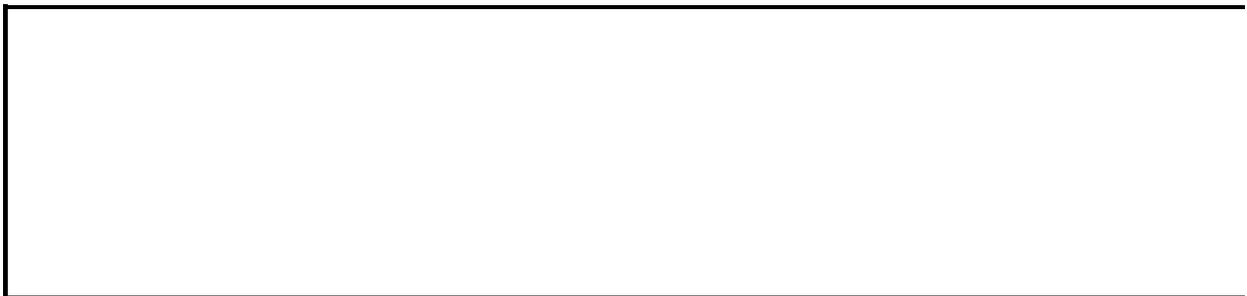
The following illustration shows the color-index values in the default color palette.



## Example

This example sets the marker background and foreground colors for the second point in series one in Chart1.

```
With Charts("Chart1").SeriesCollection(1).Points(2)  
    .MarkerBackgroundColorIndex = 4    'green  
    .MarkerForegroundColorIndex = 3    'red  
End With
```



[Show All](#)

# MarkerSize Property

Returns or sets the data-marker size, in [points](#). Read/write **Long**.

## Example

This example sets the data-marker size for all data markers on series one.

```
Worksheets(1).ChartObjects(1).Chart _  
    .SeriesCollection(1).MarkerSize = 10
```



[Show All](#)

# MarkerStyle Property

Returns or sets the marker style for a point or series in a line chart, scatter chart, or radar chart. Read/write [XlMarkerStyle](#).

XlMarkerStyle can be one of these XlMarkerStyle constants.

**xlMarkerStyleAutomatic.** Automatic markers

**xlMarkerStyleCircle.** Circular markers

**xlMarkerStyleDash.** Long bar markers

**xlMarkerStyleDiamond.** Diamond-shaped markers

**xlMarkerStyleDot.** Short bar markers

**xlMarkerStyleNone.** No markers

**xlMarkerStylePicture.** Picture markers

**xlMarkerStylePlus.** Square markers with a plus sign

**xlMarkerStyleSquare.** Square markers

**xlMarkerStyleStar.** Square markers with an asterisk

**xlMarkerStyleTriangle.** Triangular markers

**xlMarkerStyleX.** Square markers with an X

*expression*.**MarkerStyle**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example sets the marker style for series one in Chart1. The example should be run on a 2-D line chart.

```
Charts("Chart1").SeriesCollection(1) _  
    .MarkerStyle = xlMarkerStyleCircle
```



# MathCoprocesorAvailable Property

**True** if a math coprocessor is available. Read-only **Boolean**.

## Example

This example displays a message box if a math coprocessor isn't available.

```
If Not Application.MathCoprocessorAvailable Then  
    MsgBox "This macro requires a math coprocessor"  
End If
```



# Max Property

Returns or sets the maximum value of a scroll bar or spinner range. The scroll bar or spinner won't take on values greater than this maximum value. Read/write **Long**.

For information about using the **Max** worksheet function in Visual Basic, see [Using Worksheet Functions in Visual Basic](#).

## Remarks

The value of the **Max** property must be greater than the value of the **Min** property.

## Example

This example creates a scroll bar and sets its linked cell, minimum, maximum, large change, and small change values.

```
Set sb = Worksheets(1).Shapes.AddFormControl(xlScrollBar, _  
    Left:=10, Top:=10, Width:=10, Height:=200)  
With sb.ControlFormat  
    .LinkedCell = "D1"  
    .Max = 100  
    .Min = 0  
    .LargeChange = 10  
    .SmallChange = 2  
End With
```



# MaxChange Property

Returns or sets the maximum amount of change between each iteration as Microsoft Excel resolves circular references. Read/write **Double**.

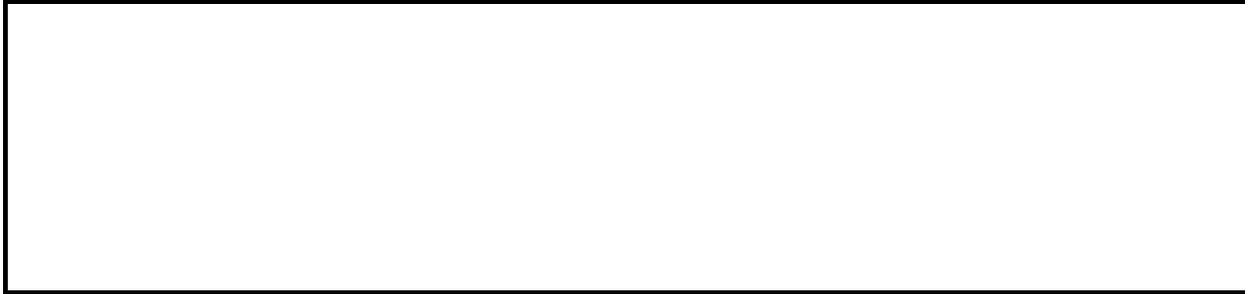
## Remarks

The [MaxIterations](#) property sets the maximum number of iterations that Microsoft Excel can use when resolving circular references.

## Example

This example sets the maximum amount of change for each iteration to 0.1.

```
Application.MaxChange = 0.1
```



# MaxCharacters Property

Returns a **Long** containing the maximum number of characters allowed in the **ListColumn** object if the **Type** property is set to **xListDataTypeText** or **xListDataTypeMultiLineText**. Returns -1 for columns whose **Type** property is set to a non-text value. Read-only **Long**.

This property is used only for lists that are linked to a SharePoint site.

*expression*.**MaxCharacters**

*expression* Required. An expression that returns a **ListDataFormat** object.

## Remarks

In Microsoft Excel, you cannot set any of the properties associated with the **ListDataFormat** object. You can set these properties, however, by modifying the list on the SharePoint site.

## Example

The following example displays the setting of the **MaxCharacters** property for the third column of a list in Sheet1 of the active workbook.

```
Dim wrksht As Worksheet
Dim objListCol As ListColumn

Set wrksht = ActiveWorkbook.Worksheets("Sheet1")
Set objListCol = wrksht.ListObjects(1).ListColumns(3)

Debug.Print objListCol.ListDataFormat.MaxCharacters
```



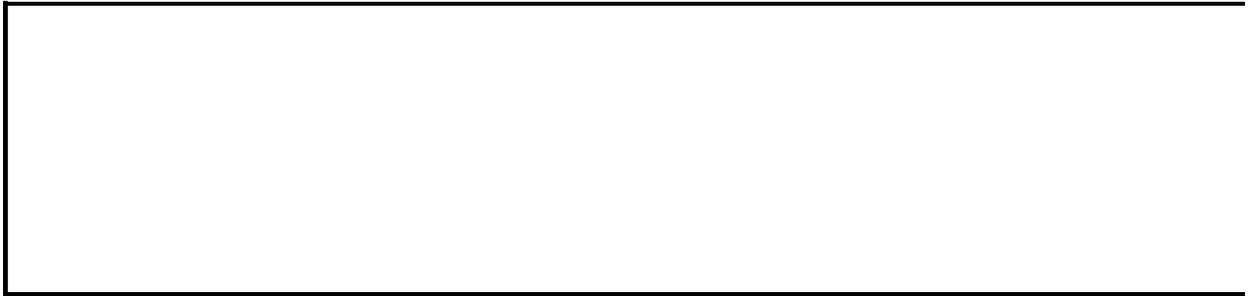
# Maximum Property

Returns or sets the maximum number of files in the list of recently used files. Can be a value from 0 (zero) through 9. Read/write **Long**.

## Example

This example sets the maximum number of files in the list of recently used files to 6.

```
Application.RecentFiles.Maximum = 6
```



# MaximumScale Property

Returns or sets the maximum value on the axis. Read/write **Double**.

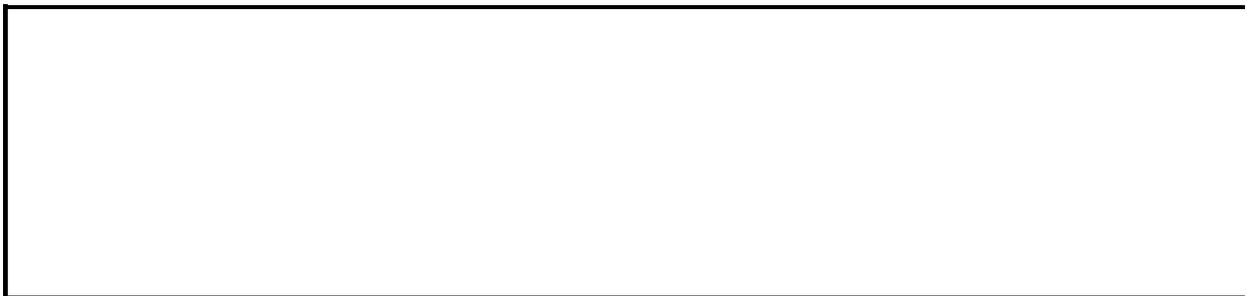
## Remarks

Setting this property sets the [MaximumScaleIsAuto](#) property to **False**.

## Example

This example sets the minimum and maximum values for the value axis in Chart1.

```
With Charts("Chart1").Axes(xlValue)  
    .MinimumScale = 10  
    .MaximumScale = 120  
End With
```



# MaximumScaleIsAuto Property

**True** if Microsoft Excel calculates the maximum value for the axis. Read/write **Boolean**.

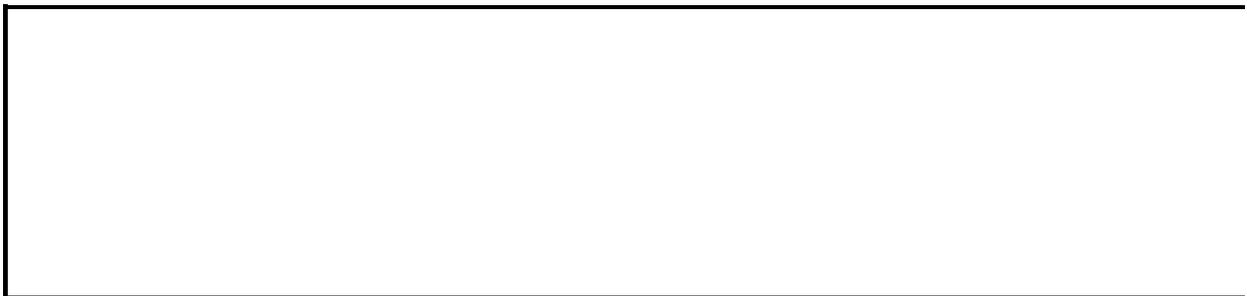
## Remarks

Setting the [MaximumScale](#) property sets this property to **False**.

## Example

This example automatically calculates the minimum scale and the maximum scale for the value axis in Chart1.

```
With Charts("Chart1").Axes(xlValue)  
    .MinimumScaleIsAuto = True  
    .MaximumScaleIsAuto = True  
End With
```



# MaxIterations Property

Returns or sets the maximum number of iterations that Microsoft Excel can use to resolve a circular reference. Read/write **Long**.

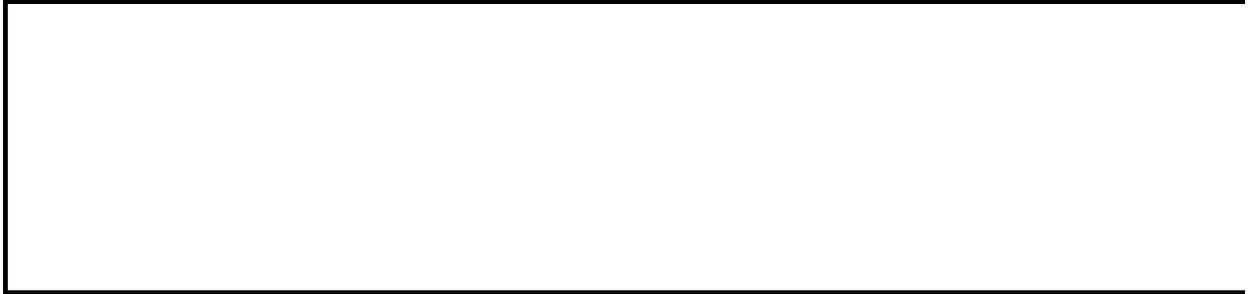
## Remarks

The [MaxChange](#) property sets the maximum amount of change between each iteration when Microsoft Excel is resolving circular references.

## Example

This example sets the maximum number of iterations at 1000.

```
Application.MaxIterations = 1000
```



# MaxNumber Property

Returns a **Variant** containing the maximum value allowed in this field in the list column. The **Nothing** object is returned if a maximum value number has not been specified or if the **Type** property setting is such that a maximum value for the column is not applicable. Read-only **Variant**.

This property is used only for lists that are linked to a SharePoint site.

*expression*.**MaxNumber**

*expression* Required. An expression that returns a **ListDataFormat** object.

## Remarks

In Microsoft Excel, you cannot set any of the properties associated with the **ListDataFormat** object. You can set these properties, however, by modifying the list on the SharePoint site.

## Example

The following example displays the setting of the **MaxNumber** property for the third column of a list in Sheet1 of the active workbook.

```
Dim wrksht As Worksheet
Dim objListCol As ListColumn

Set wrksht = ActiveWorkbook.Worksheets("Sheet1")
Set objListCol = wrksht.ListObjects(1).ListColumns(3)

Debug.Print objListCol.ListDataFormat.MaxNumber
```



[Show All](#)

# MDX Property

Returns a **String** indicating the MDX (Multidimensional Expression) that would be sent to the provider to populate the current PivotTable view. Read-only.

*expression*.**MDX**

*expression* Required. An expression that returns a [PivotTable](#) object.

## Remarks

Querying this value for a non-[Online Analytical Processing \(OLAP\)](#) PivotTable, or when there is no PivotTable view (no data items), will return a run-time error.

## Example

This example returns the MDX string for the PivotTable. It assumes a PivotTable exists on the active worksheet.

```
Sub CheckMDX()  
    Dim pvtTable As PivotTable  
    Set pvtTable = ActiveSheet.PivotTables(1)  
    MsgBox "The MDX string for the PivotTable is: " & _  
        pvtTable.MDX  
End Sub
```



# MemoryFree Property

Returns the amount of memory that's still available for Microsoft Excel to use, in bytes. Read-only **Long**.

## Example

This example displays a message box showing the number of free bytes.

```
MsgBox "Microsoft Excel has " & _  
Application.MemoryFree & " bytes free"
```



# MemoryUsed Property

**Application** object: Returns the amount of memory that Microsoft Excel is currently using, in bytes. Read-only **Long**.

**PivotCache** or **PivotField** object: Returns the amount of memory currently being used by the object, in bytes. Read-only **Long**.

## Remarks

For **PivotCache** objects, this property reflects the transient state of the cache at the time that it's queried.

If the **PivotCache** object has no PivotTable report attached to it, this property returns 0 (zero).

## Example

This example displays a message box showing the number of bytes that Microsoft Excel is currently using.

```
MsgBox "Microsoft Excel is currently using " & _  
    Application.MemoryUsed & " bytes"
```



# MergeArea Property

Returns a **Range** object that represents the merged range containing the specified cell. If the specified cell isn't in a merged range, this property returns the specified cell. Read-only **Variant**.

*expression*.**MergeArea**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

The **MergeArea** property only works on a single-cell range.

## Example

This example sets the value of the merged range that contains cell A3.

```
Set ma = Range("a3").MergeArea
If ma.Address = "$A$3" Then
    MsgBox "not merged"
Else
    ma.Cells(1, 1).Value = "42"
End If
```



# MergeCells Property

**True** if the range or style contains merged cells. Read/write **Variant**.

## Remarks

When you select a range that contains merged cells, the resulting selection may be different from the intended selection. Use the **Address** property to check the address of the selected range.

## Example

This example sets the value of the merged range that contains cell A3.

```
Set ma = Range("a3").MergeArea
If Range("a3").MergeCells Then
    ma.Cells(1, 1).Value = "42"
End If
```



# MergeLabels Property

**True** if the specified PivotTable report's outer-row item, column item, subtotal, and grand total labels use merged cells. Read/write **Boolean**.

## Example

This example causes the first PivotTable report on worksheet one to use merged-cell outer-row item, column item, subtotal, and grand total labels.

```
Worksheets(1).PivotTables(1).MergeLabels = True
```



# Message Property

Returns or sets the message text for the routing slip. This text is used as the body text of mail messages that are used to route the workbook. Read/write **String**.

## Example

This example sends Book1.xls to three recipients, one after another.

```
Workbooks("BOOK1.XLS").HasRoutingSlip = True
With Workbooks("BOOK1.XLS").RoutingSlip
    .Delivery = xlOneAfterAnother
    .Recipients = Array("Adam Bendel", _
        "Jean Selva", "Bernard Gabor")
    .Subject = "Here is BOOK1.XLS"
    .Message = "Here is the workbook. What do you think?"
End With
Workbooks("BOOK1.XLS").Route
```



# Min Property

Returns or sets the minimum value of a scroll bar or spinner range. The scroll bar or spinner won't take on values less than this minimum value. Read/write **Long**.

For information about using the **Min** worksheet function in Visual Basic, see [Using Worksheet Functions in Visual Basic](#).

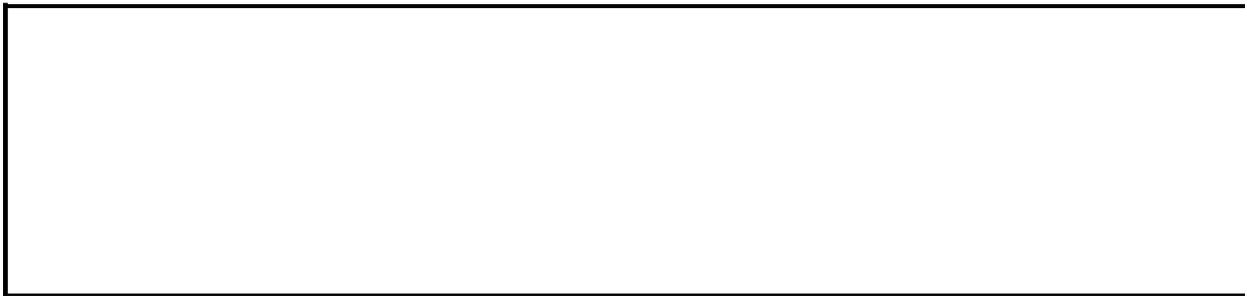
## Remarks

The value of the **Min** property must be less than the value of the **Max** property.

## Example

This example creates a scroll bar and sets its linked cell, minimum, maximum, large change, and small change values.

```
Set sb = Worksheets(1).Shapes.AddFormControl(xlScrollBar, _  
    Left:=10, Top:=10, Width:=10, Height:=200)  
With sb.ControlFormat  
    .LinkedCell = "D1"  
    .Max = 100  
    .Min = 0  
    .LargeChange = 10  
    .SmallChange = 2  
End With
```



# MinimumScale Property

Returns or sets the minimum value on the axis. Read/write **Double**.

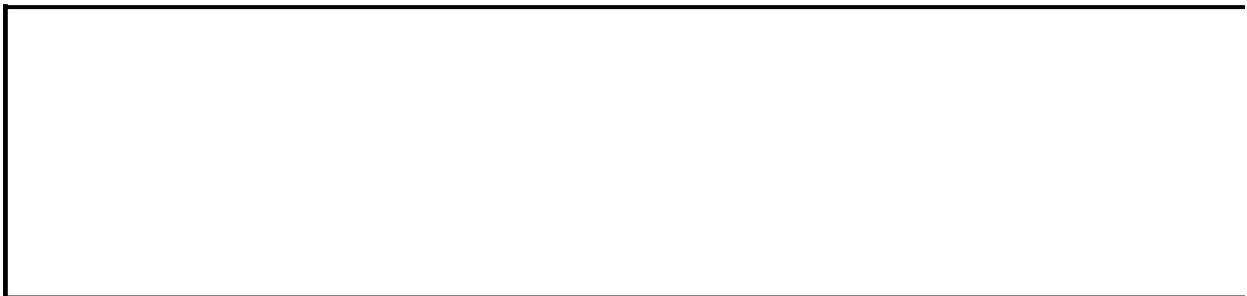
## Remarks

Setting this property sets the [MinimumScaleIsAuto](#) property to **False**.

## Example

This example sets the minimum and maximum values for the value axis in Chart1.

```
With Charts("Chart1").Axes(xlValue)  
    .MinimumScale = 10  
    .MaximumScale = 120  
End With
```



# MinimumScaleIsAuto Property

**True** if Microsoft Excel calculates the minimum value for the axis. Read/write **Boolean**.

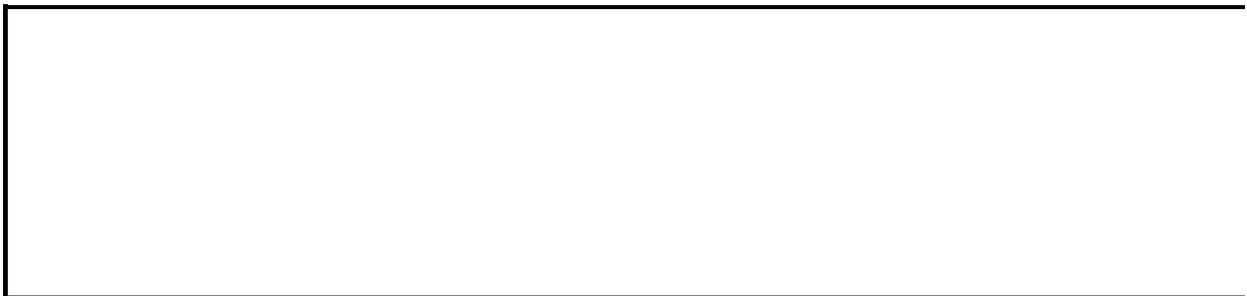
## Remarks

Setting the [MinimumScale](#) property sets this property to **False**.

## Example

This example automatically calculates the minimum scale and the maximum scale for the value axis in Chart1.

```
With Charts("Chart1").Axes(xlValue)  
    .MinimumScaleIsAuto = True  
    .MaximumScaleIsAuto = True  
End With
```



# MinNumber Property

Returns a **Variant** containing the minimum value allowed in this field in the list column. This can be a negative floating point number. This property will return the **Nothing** object if no value has been specified for this field or if the setting of the **Type** property is such that a minimum value is not applicable to the column. Read-only **Variant**.

This property is used only for lists that are linked to a SharePoint site.

*expression*.**MinNumber**

*expression* Required. An expression that returns a **ListDataFormat** object.

## Remarks

In Microsoft Excel, you cannot set any of the properties associated with the **ListDataFormat** object. You can set these properties, however, by modifying the list on the SharePoint site.

## Example

The following example displays the setting of the **MinNumber** property for the third column of a list in Sheet1 of the active workbook.

```
Dim wrksht As Worksheet
Dim objListCol As ListColumn

Set wrksht = ActiveWorkbook.Worksheets("Sheet1")
Set objListCol = wrksht.ListObjects(1).ListColumns(3)

Debug.Print objListCol.ListDataFormat.MinNumber
```



# MinorGridlines Property

Returns a [Gridlines](#) object that represents the minor gridlines for the specified axis. Only axes in the primary axis group can have gridlines. Read-only.

## Example

This example sets the color of the minor gridlines for the value axis in Chart1.

```
With Charts("Chart1").Axes(xlValue)
    If .HasMinorGridlines Then
        .MinorGridlines.Border.ColorIndex = 5    'set color to blue
    End If
End With
```



[Show All](#)

# MinorTickMark Property

Returns or sets the type of minor tick mark for the specified axis. Read/write [xlTickMark](#).

XlTickMark can be one of these XlTickMark constants.

**xlTickMarkInside**

**xlTickMarkOutside**

**xlTickMarkCross**

**xlTickMarkNone**

*expression*.**MinorTickMark**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example sets the minor tick marks for the value axis in Chart1 to be inside the axis.

```
Charts("Chart1").Axes(xlValue).MinorTickMark = xlTickMarkInside
```



# MinorUnit Property

Returns or sets the minor units on the axis. Read/write **Double**.

## Remarks

Setting this property sets the [MinorUnitIsAuto](#) property to **False**.

Use the [TickMarkSpacing](#) property to set tick mark spacing on the category axis.

## Example

This example sets the major and minor units for the value axis in Chart1.

```
With Charts("Chart1").Axes(xlValue)  
    .MajorUnit = 100  
    .MinorUnit = 20  
End With
```



# MinorUnitIsAuto Property

**True** if Microsoft Excel calculates minor units for the axis. Read/write **Boolean**.

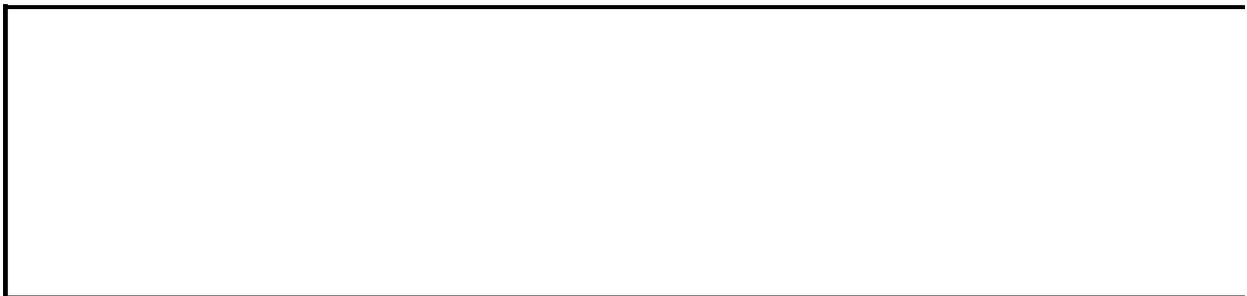
## Remarks

Setting the [MinorUnit](#) property sets this property to **False**.

## Example

This example automatically calculates major and minor units for the value axis in Chart1.

```
With Charts("Chart1").Axes(xlValue)  
    .MajorUnitIsAuto = True  
    .MinorUnitIsAuto = True  
End With
```



[Show All](#)

# MinorUnitScale Property

Returns or sets the minor unit scale value for the category axis when the **CategoryType** property is set to **xlTimeScale**. Read/write [xlTimeUnit](#).

XlTimeUnit can be one of these XlTimeUnit constants.

**xlMonths**

**xlDays**

**xlYears**

*expression*.**MinorUnitScale**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example sets the category axis to use a time scale and sets the major and minor units.

```
With Charts(1).Axes(xlCategory)
    .CategoryType = xlTimeScale
    .MajorUnit = 5
    .MajorUnitScale = xlDays
    .MinorUnit = 1
    .MinorUnitScale = xlDays
End With
```



[Show All](#)

# MissingItemsLimit Property

Returns or sets the maximum quantity of unique items per PivotTable field that are retained even when they have no supporting data in the cache records.

Read/write [XLPivotTableMissingItems](#).

XLPivotTableMissingItems can be one of these XLPivotTableMissingItems constants.

**xlMissingItemsDefault** The default number of unique items per PivotField allowed.

**xlMissingItemsMax** The maximum number of unique items per PivotField allowed (32,500).

**xlMissingItemsNone** No unique items per PivotField allowed (zero).

*expression*.**MissingItemsLimit**

*expression* Required. An expression that returns a [PivotCache](#) object.

## Remarks

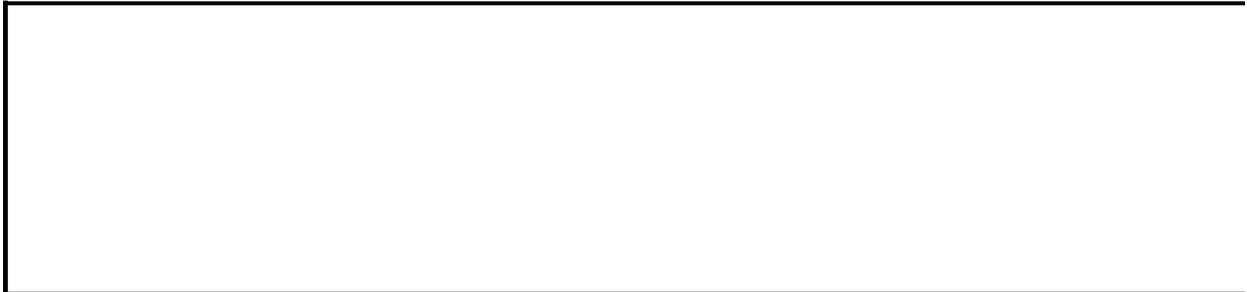
This property can be set to a value between 0 and 32500. If an integer less than zero is specified, this is equivalent to specifying **xlMissingItemsDefault**. Integers greater than 32,500 can be specified but will have the same effect as specifying **xlMissingItemsMax**.

The **MissingItemsLimit** property only works for non-OLAP PivotTables; otherwise, a run-time error can occur.

## Example

This example determines the maximum quantity of unique items per field and notifies the user. The example assumes a PivotTable exists on the active worksheet.

```
Sub CheckMissingItemsList()  
  
    Dim pvtCache As PivotCache  
  
    Set pvtCache = Application.ActiveWorkbook.PivotCaches.Item(1)  
  
    ' Determine the maximum number of unique items allowed per Pivot  
    Select Case pvtCache.MissingItemsLimit  
        Case xlMissingItemsDefault  
            MsgBox "The default value of unique items per PivotField  
        Case xlMissingItemsMax  
            MsgBox "The maximum value of unique items per PivotField  
        Case xlMissingItemsNone  
            MsgBox "No unique items per PivotField are allowed."  
    End Select  
  
End Sub
```



# MouseAvailable Property

**True** if a mouse is available. Read-only **Boolean**.

## Example

This example displays a message if a mouse isn't available.

```
If Application.MouseAvailable = False Then  
    MsgBox "Your system does not have a mouse"  
End If
```



# MoveAfterReturn Property

**True** if the active cell will be moved as soon as the ENTER (RETURN) key is pressed. Read/write **Boolean**.

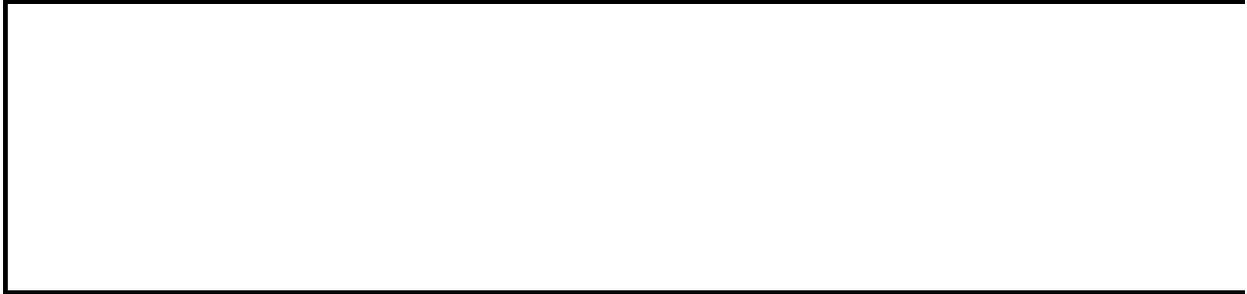
## Remarks

Use the [MoveAfterReturnDirection](#) property to specify the direction in which the active cell is to be moved.

## Example

This example sets the **MoveAfterReturn** property to **True**.

```
Application.MoveAfterReturn = True
```



[Show All](#)

# MoveAfterReturnDirection Property

Returns or sets the direction in which the active cell is moved when the user presses ENTER. Read/write [XIDirection](#).

XIDirection can be one of these XIDirection constants.

**xIDown**

**xlToLeft**

**xlToRight**

**xlUp**

*expression*.**MoveAfterReturnDirection**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

If the [MoveAfterReturn](#) property is **False**, the selection doesn't move at all, regardless of how the **MoveAfterReturnDirection** property is set.

## Example

This example causes the active cell to move to the right when the user presses ENTER.

```
Application.MoveAfterReturn = True  
Application.MoveAfterReturnDirection = xlToRight
```



# MultiSelect Property

Returns or sets the selection mode of the specified list box. Can be one of the following constants: **xlNone**, **xlSimple**, or **xlExtended**. Read/write **Long**.

## Remarks

Single select (**xlNone**) allows only one item at a time to be selected. Clicking the mouse or pressing the SPACEBAR cancels the selection and selects the clicked item.

Simple multiselect (**xlSimple**) toggles the selection on an item in the list when click it with the mouse or press the SPACEBAR when the focus is on the item. This mode is appropriate for pick lists, in which there are often multiple items selected.

Extended multiselect (**xlExtended**) usually acts like a single-selection list box, so when you click an item, you cancel all other selections. When you hold down SHIFT while clicking the mouse or pressing an arrow key, you select items sequentially from the current item. When you hold down CTRL while clicking the mouse, you add single items to the list. This mode is appropriate when multiple items are allowed but not often used.

You can use the **Value** or **ListIndex** property to return and set the selected item in a single-select list box.

You cannot link multiselect list boxes by using the **LinkedCell** property.

## Example

This example creates a simple multiselect list box.

```
Set lb = Worksheets(1).Shapes.AddFormControl(xlListBox, _  
    Left:=10, Top:=10, Height:=100, Width:100)  
lb.ControlFormat.MultiSelect = xlSimple
```



# MultiUserEditing Property

**True** if the workbook is open as a shared list. Read-only **Boolean**.

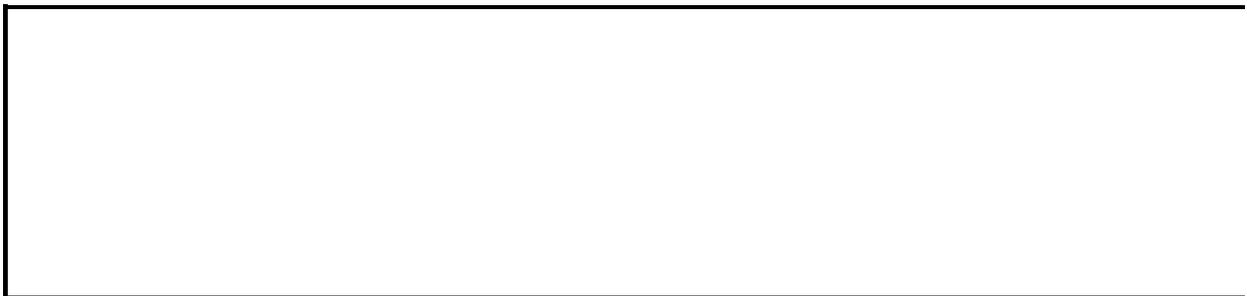
## Remarks

To save a workbook as a shared list, use the **SaveAs** method. To switch the workbook from shared mode to exclusive mode, use the **ExclusiveAccess** method.

## Example

This example determines whether the active workbook is open in exclusive mode. If it is, the example saves the workbook as a shared list.

```
If Not ActiveWorkbook.MultiUserEditing Then  
    ActiveWorkbook.SaveAs fileName:=ActiveWorkbook.FullName, _  
        accessMode:=xlShared  
End If
```



[Show All](#)

# Name Property

Name property as it applies to the [Chart](#), [ChartObject](#), [ColorFormat](#), [CustomProperty](#), [Name](#), [OLEObject](#), [Parameter](#), [PivotField](#), [PivotItem](#), [PivotTable](#), [QueryTable](#), [Scenario](#), [Series](#), [Shape](#), [ShapeRange](#), [Trendline](#), and [Worksheet](#) objects.

Returns or sets the name of the object. Read/write **String**.

*expression*.**Name**

*expression* Required. An expression that returns one of the above objects.

Name property as it applies to the [AddIn](#), [Application](#), [AxisTitle](#), [CalculatedMember](#), [ChartArea](#), [ChartTitle](#), [Corners](#), [CubeField](#), [CustomView](#), [DataLabel](#), [DataLabels](#), [DisplayUnitLabel](#), [DownBars](#), [DropLines](#), [ErrorBars](#), [Floor](#), [Gridlines](#), [HiLoLines](#), [Hyperlink](#), [Legend](#), [PlotArea](#), [RecentFile](#), [SeriesLines](#), [SmartTag](#), [SmartTagAction](#), [Style](#), [TickLabels](#), [UpBars](#), [Walls](#), and [Workbook](#) objects.

Returns the name of the object. Read-only **String**.

*expression*.**Name**

*expression* Required. An expression that returns one of the above objects.

Name property as it applies to the [Font](#) and [Range](#) objects.

Returns or sets the name of the object. The name of a **Range** object is a [Name](#) object. For every other type of object, the name is a **String**. Read/write **Variant**.

*expression*.**Name**

*expression* Required. An expression that returns one of the above objects.

Name property as it applies to the [ListColumn](#) object.

Returns or sets the name of the list column. This is also used as the display name of the list column. This name must be unique within the list. Read/write **String**.

*expression*.**Name**

*expression* Required. An expression that returns a **ListColumn** object.

**Note** If this list is linked to a SharePoint list, this property is read-only.

[Name property as it applies to the \*\*ListObject\*\* object.](#)

Returns or sets the name of the **ListObject** object. This name is used solely as a unique identifier for the **Item** property of the **ListObjects** collection objects. This property can only be set through the object model. Read/write **String**.

*expression*.**Name**

*expression* Required. An expression that returns an **ListObject** object.

## Remarks

By default, each **ListObject** object name begins with the word "List", followed by a number (no spaces). If an attempt is made to set the **Name** property to a name already used by another **ListObject** object, a run-time error is thrown.

[Name property as it applies to the \*\*XmlMap\*\* object.](#)

Returns or sets the friendly name used to uniquely identify a mapping in the workbook. Read/write **String**.

*expression*.**Name**

*expression* Required. An expression that returns an **XmlMap** object.

## Remarks

The string specified for the **Name** property must be unique within the workbook, and cannot exceed 255 characters.

## Remarks

The following table shows example values of the **Name** property and related properties given an [OLAP](#) data source with the unique name "[Europe].[France].[Paris]" and a non-OLAP data source with the item name "Paris".

<b>Property</b>	<b>Value (OLAP data source)</b>	<b>Value (non-OLAP data source)</b>
<b>Caption</b>	Paris	Paris
<b>Name</b>	[Europe].[France].[Paris] (read-only)	Paris
<b>SourceName</b>	[Europe].[France].[Paris] (read-only)	(Same as SQL property value, read-only)
<b>Value</b>	[Europe].[France].[Paris] (read-only)	Paris

When specifying an index into the [PivotItems](#) collection, you can use the syntax shown in the following table.

<b>Syntax (OLAP data source)</b>	<b>Syntax (non-OLAP data source)</b>
expression.PivotItems("[Europe].[France].[Paris]")	expression.PivotItems("Paris")

When using the [Item](#) property to reference a specific member of a collection, you can use the text index name as shown in the following table.

<b>Name (OLAP data source)</b>	<b>Name (non-OLAP data source)</b>
[Europe].[France].[Paris]	Paris

## Example

This example displays the name of style one in the active workbook, first in the language of the macro and then in the language of the user.

```
With ActiveWorkbook.Styles(1)
    MsgBox "The name of the style: " & .Name
    MsgBox "The localized name of the style: " & .NameLocal
End With
```

The following example displays the name of the default **ListObject** object in sheet1 of the active workbook.

```
Sub Test
    Dim wrksht As Worksheet
    Dim oListObj As ListObject

    Set wrksht = ActiveWorkbook.Worksheets("Sheet1")
    Set oListObj = wrksht.ListObjects(1)

    MsgBox oListObj.Name
End Sub
```



# NameIsAuto Property

**True** if Microsoft Excel automatically determines the name of the trendline.  
Read/write **Boolean**.

## Example

This example sets Microsoft Excel to automatically determine the name for trendline one in Chart1. The example should be run on a 2-D column chart that contains a single series with a trendline.

```
Charts("Chart1").SeriesCollection(1) _  
    .Trendlines(1).NameIsAuto = True
```



# NameLocal Property

Returns or sets the name of the object, in the language of the user. Read/write **String** for **Name**, read-only **String** for **Style**.

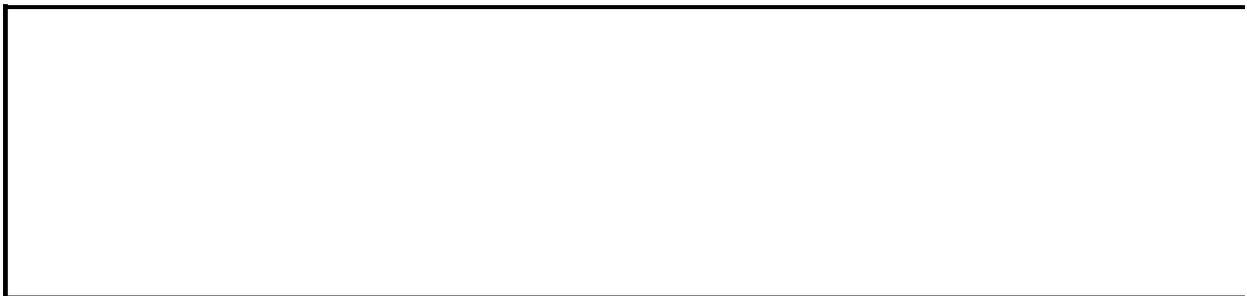
## Remarks

If the style is a built-in style, this property returns the name of the style in the language of the current locale.

## Example

This example displays the name and localized name of style one in the active workbook.

```
With ActiveWorkbook.Styles(1)
    MsgBox "The name of the style is " & .Name
    MsgBox "The localized name of the style is " & .NameLocal
End With
```



# Names Property

For an **Application** object, returns a [Names](#) collection that represents all the names in the active workbook. For a **Workbook** object, returns a **Names** collection that represents all the names in the specified workbook (including all worksheet-specific names). For a **Worksheet** object, returns a **Names** collection that represents all the worksheet-specific names (names defined with the "WorksheetName!" prefix). Read-only **Names** object.

*expression*.**Names**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

For information about returning a single member of a collection, see [Returning an Object from a Collection](#).

Using this property without an object qualifier is equivalent to using `ActiveWorkbook.Names`.

## Example

This example defines the name "myName" for cell A1 on Sheet1.

```
ActiveWorkbook.Names.Add Name:="myName", RefersToR1C1:= _  
    "=Sheet1!R1C1"
```



[Show All](#)

# Namespace Property

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Returns an [XmlNamespace](#) object that represents the target [namespace](#) for the specified schema. Read-only.

*expression*.**Namespace**

*expression* Required. An expression that returns an [XmlSchema](#) object.



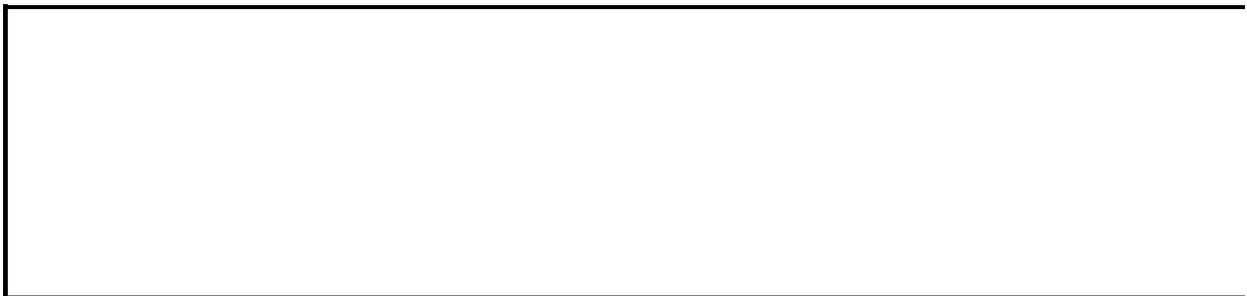
# Native Property

Returns a provider-specific numeric value that specifies an error. The error number corresponds to an error condition that resulted after the most recent OLE DB query. Read-only **Long**.

## Example

This example displays the native error number and other error information returned by the most recent OLE DB query.

```
Set objEr = Application.OLEDBErrors(1)
MsgBox "The following error occurred:" & _
    objEr.Number & ", " & objEr.Native & ", " & _
    objEr.ErrorString & " : " & objEr.SqlState
```



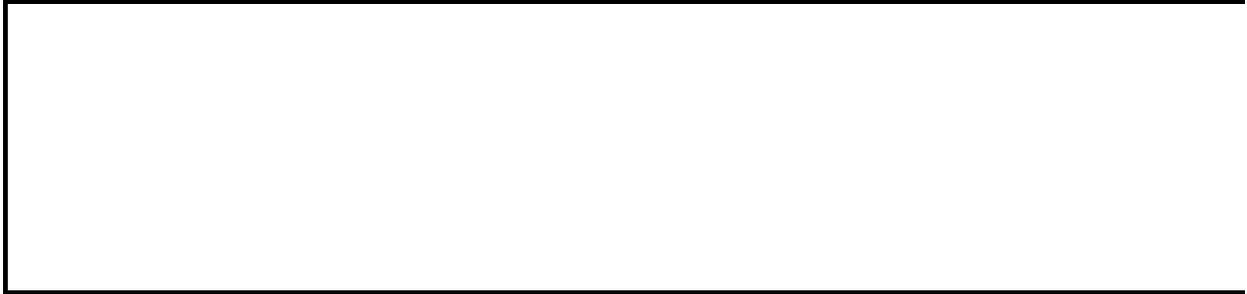
# NetworkTemplatesPath Property

Returns the network path where templates are stored. If the network path doesn't exist, this property returns an empty string. Read-only **String**.

## Example

This example displays the network path where templates are stored.

Msgbox Application.**NetworkTemplatesPath**



# NewWorkbook Property

Returns a **NewFile** object.

*expression*.**NewWorkbook**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel sets the variable `wkbOne` to a **NewFile** object.

```
Sub SetStartWorking()  
    Dim wkbOne As NewFile  
    ' Create a reference to an instance of the NewFile object.  
    Set wkbOne = Application.NewWorkbook  
End Sub
```



# Next Property

Returns a [Chart](#), [Range](#), or [Worksheet](#) object that represents the next sheet or cell. Read-only.

## Remarks

If the object is a range, this property emulates the TAB key, although the property returns the next cell without selecting it.

On a protected sheet, this property returns the next unlocked cell. On an unprotected sheet, this property always returns the cell immediately to the right of the specified cell.

## Example

This example selects the next unlocked cell on Sheet1. If Sheet1 is unprotected, this is the cell immediately to the right of the active cell.

```
Worksheets("Sheet1").Activate  
ActiveCell.Next.Select
```



[Show All](#)

# Nodes Property

 [Nodes property as it applies to the \*\*Diagram\*\* object.](#)

Returns a **DiagramNodes** object that contains a flat list of all the nodes in the specified diagram.

*expression.Nodes*

*expression* Required. An expression that returns a **Diagram** object.

 [Nodes property as it applies to the \*\*Shape\*\* and \*\*ShapeRange\*\* objects.](#)

Returns a **ShapeNodes** collection that represents the geometric description of the specified shape. Applies to **Shape** or **ShapeRange** objects that represent freeform drawings.

*expression.Nodes*

*expression* Required. An expression that returns one of the above objects.

## Example

 [As it applies to the \*\*Shape\*\* and \*\*ShapeRange\*\* objects.](#)

This example adds a smooth node with a curved segment after node four in shape three on myDocument. Shape three must be a freeform drawing with at least four nodes.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(3).Nodes
    .Insert 4, msoSegmentCurve, msoEditingSmooth, 210, 100
End With
```



[Show All](#)

# NormalizedHeight Property

**True** if all characters (both uppercase and lowercase) in the specified WordArt are the same height. Read/write [MsoTriState](#).

MsoTriState can be one of these MsoTriState constants.

**msoCTrue**

**msoFalse**

**msoTriStateMixed**

**msoTriStateToggle**

**msoTrue** All characters (both uppercase and lowercase) in the specified WordArt are the same height.

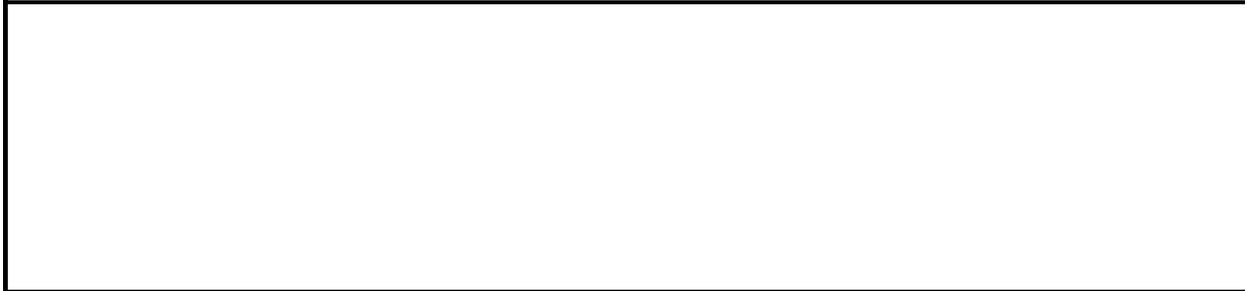
*expression*.**NormalizedHeight**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example adds WordArt that contains the text "Test Effect" to myDocument and gives the new WordArt the name "texteff1." The code then makes all characters in the shape named "texteff1" the same height.

```
Set myDocument = Worksheets(1)
myDocument.Shapes.AddTextEffect( _
    PresetTextEffect:=msoTextEffect1, _
    Text:="Test Effect", FontName:="Courier New", _
    FontSize:=44, FontBold:=True, _
    FontItalic:=False, Left:=10, Top:=10).Name = "texteff1"
myDocument.Shapes("texteff1").TextEffect.NormalizedHeight = msoTrue
```



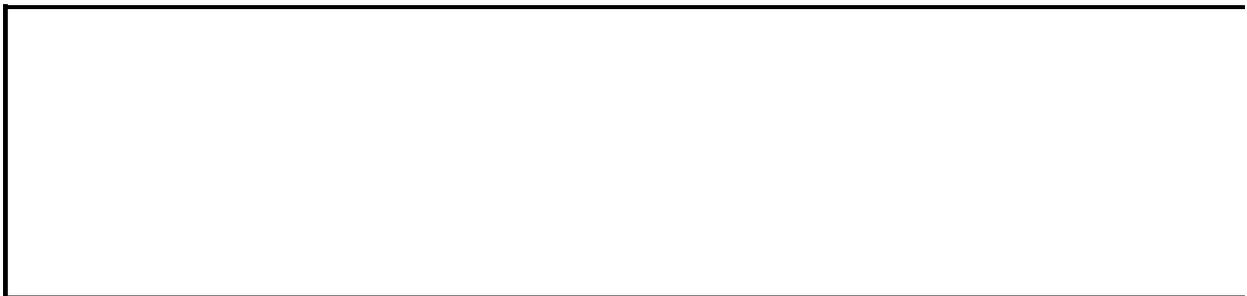
# NullString Property

Returns or sets the string displayed in cells that contain null values when the **DisplayNullString** property is **True**. The default value is an empty string ("").  
Read/write **String**.

## Example

This example causes the PivotTable report to display "NA" in cells that contain null values.

```
With Worksheets(1).PivotTables("Pivot1")  
    .NullString = "NA"  
    .DisplayNullString = True  
End With
```



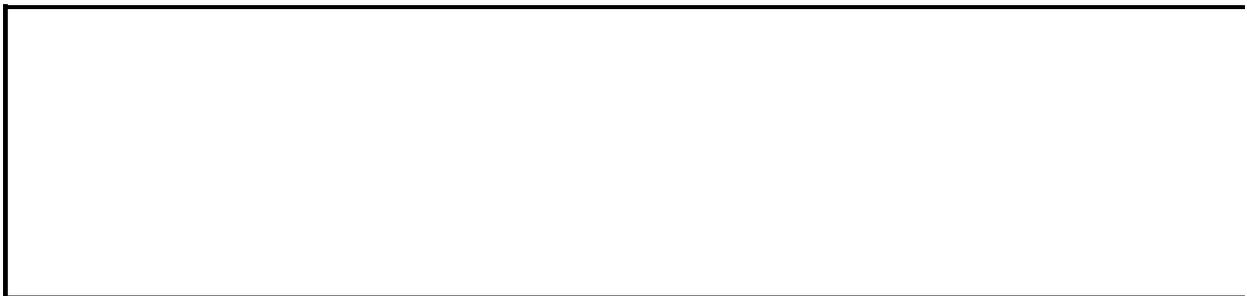
# Number Property

Returns a numeric value that specifies an error. The error number corresponds to a unique trap number corresponding to an error condition that resulted after the most recent OLE DB query. Read-only **Long**.

## Example

This example displays the error number and other error information returned by the most recent OLE DB query.

```
Set objEr = Application.OLEDBErrors(1)
MsgBox "The following error occurred:" & _
    objEr.Number & ", " & objEr.Native & ", " & _
    objEr.ErrorString & " : " & objEr.SqlState
```



# NumberAsText Property

When set to **True** (default), Microsoft Excel identifies, with an **AutoCorrect Options** button, selected cells that contain numbers written as text. **False** disables error checking for numbers written as text. Read/write **Boolean**.

*expression*.**NumberAsText**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In the following example, the **AutoCorrect Options** button appears for cell A1, which contains a number stored as text.

```
Sub CheckNumberAsText()  
    ' Simulate an error by referencing a number stored as text.  
    Application.ErrorCheckingOptions.NumberAsText = True  
    Range("A1").Value = "'1"  
End Sub
```



[Show All](#)

# NumberFormat Property

 [NumberFormat property as it applies to the \*\*DataLabel\*\*, \*\*DataLabels\*\*, \*\*PivotField\*\*, \*\*Style\*\*, and \*\*TickLabels\*\* objects.](#)

Returns or sets the format code for the object. Read/write **String**.

*expression*.**NumberFormat**

*expression* Required. An expression that returns one of the above objects.

 [NumberFormat property as it applies to the \*\*CellFormat\*\* and \*\*Range\*\* objects.](#)

Returns or sets the format code for the object. Returns **Null** if all cells in the specified range don't have the same number format. Read/write **Variant**.

*expression*.**NumberFormat**

*expression* Required. An expression that returns one of the above objects.

## Remarks

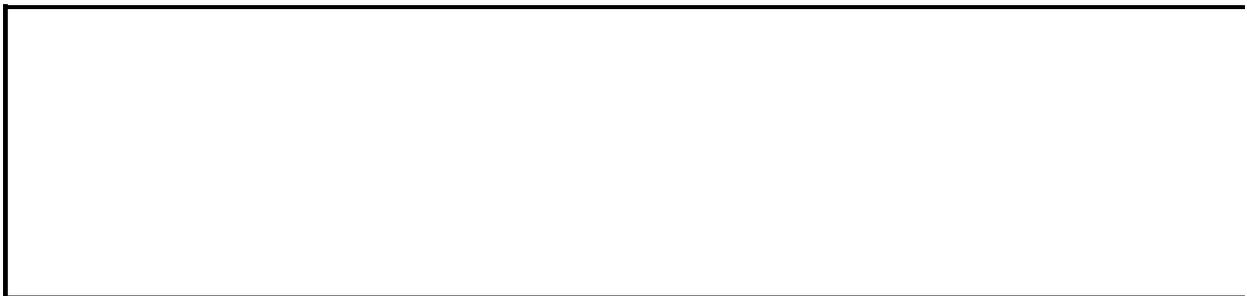
For the **PivotField** object, you can set the **NumberFormat** property only for a data field.

The format code is the same string as the **Format Codes** option in the **Format Cells** dialog box. The **Format** function uses different format code strings than do the **NumberFormat** and **NumberFormatLocal** properties.

## Example

These examples set the number format for cell A17, row one, and column C (respectively) on Sheet1.

```
Worksheets("Sheet1").Range("A17").NumberFormat = "General"  
Worksheets("Sheet1").Rows(1).NumberFormat = "hh:mm:ss"  
Worksheets("Sheet1").Columns("C"). _  
    NumberFormat = "$#,##0.00_);[Red]($#,##0.00)"
```



# NumberFormatLinked Property

**True** if the number format is linked to the cells (so that the number format changes in the labels when it changes in the cells). Read/write **Boolean**.

## Example

This example links the number format for tick-mark labels to its cells for the value axis in Chart1.

```
Charts("Chart1").Axes(xlValue).TickLabels.NumberFormatLinked = True
```



[Show All](#)

# NumberFormatLocal Property

 [NumberFormatLocal property as it applies to the \*\*Style\*\* object.](#)

Returns or sets the format code for the object as a string in the language of the user. Read/write **String**.

*expression*.**NumberFormatLocal**

*expression* Required. An expression that returns a **Style** object.

 [NumberFormatLocal property as it applies to the \*\*CellFormat\*\*, \*\*DataLabel\*\*, \*\*DataLabels\*\*, \*\*Range\*\*, and \*\*TickLabels\*\* objects.](#)

Returns or sets the format code for the object as a string in the language of the user. Read/write **Variant**.

*expression*.**NumberFormatLocal**

*expression* Required. An expression that returns one of the above objects.

## Remarks

The **Format** function uses different format code strings than do the **NumberFormat** and **NumberFormatLocal** properties.

## Example

[As it applies to the `CellFormat`, `DataLabel`, `DataLabels`, `Range`, and `TickLabels` objects.](#)

This example displays the number format for cell A1 on Sheet1 in the language of the user.

```
MsgBox "The number format for cell A1 is " & _  
    Worksheets("Sheet1").Range("A1").NumberFormatLocal
```



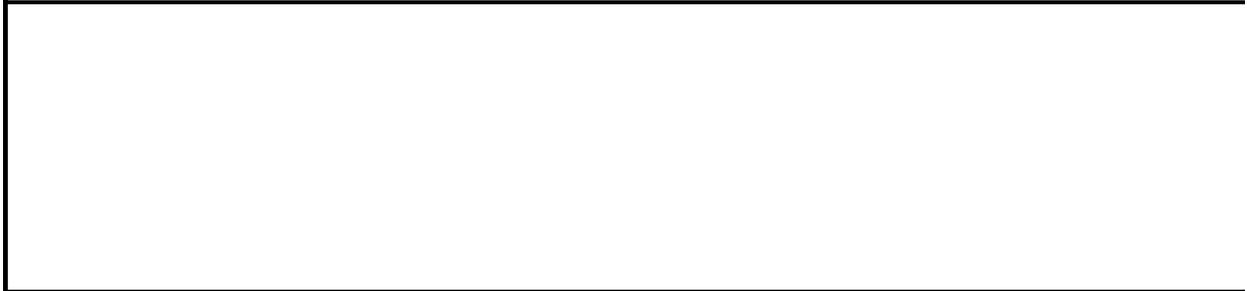
# Object Property

Returns the OLE Automation object associated with this OLE object. Read-only **Object**.

## Example

This example inserts text at the beginning of an embedded Word document object on Sheet1. Note that the three statements in the **With** control structure are WordBasic statements.

```
Set wordObj = Worksheets("Sheet1").OLEObjects(1)
wordObj.Activate
With wordObj.Object.Application.WordBasic
    .StartOfDocument
    .Insert "This is the beginning"
    .InsertPara
End With
```



[Show All](#)

# Obscured Property

**True** if the shadow of the specified shape appears filled in and is obscured by the shape, even if the shape has no fill. **False** if the shadow has no fill and the outline of the shadow is visible through the shape if the shape has no fill.

Read/write [MsoTriState](#).

MsoTriState can be one of these MsoTriState constants.

**msoCTrue**

**msoFalse** The shadow has no fill and the outline of the shadow is visible through the shape if the shape has no fill.

**msoTriStateMixed**

**msoTriStateToggle**

**msoTrue** The shadow of the specified shape appears filled in and is obscured by the shape, even if the shape has no fill.

*expression*.**Obscured**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example sets the horizontal and vertical offsets for the shadow of shape three on myDocument. The shadow is offset 5 points to the right of the shape and 3 points above it. If the shape doesn't already have a shadow, this example adds one to it. The shadow will be filled in and obscured by the shape, even if the shape has no fill.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(3).Shadow
    .Visible = True
    .OffsetX = 5
    .OffsetY = -3
    .Obscured = msoTrue
End With
```



# ODBCErrors Property

Returns an [ODBCErrors](#) collection that contains all the ODBC errors generated by the most recent query table or PivotTable report operation. Read-only.

For more information about returning a single object from a collection, see [Returning an Object from a Collection](#).

## Remarks

If there's more than one query running at the same time, the **ODBCErrors** collection contains the ODBC errors from the query that's finished last.

## Example

This example refreshes query table one and displays any ODBC errors that occur.

```
With Worksheets(1).QueryTables(1)
    .Refresh
    Set errs = Application.ODBCErrors
    If errs.Count > 0 Then
        Set r = .Destination.Cells(1)
        r.Value = "The following errors occurred:"
        c = 0
        For Each er In errs
            c = c + 1
            r.Offset(c, 0).value = er.ErrorString
            r.Offset(c, 1).value = er.SqlState
        Next
    Else
        MsgBox "Query complete: all records returned."
    End If
End With
```



# ODBCTimeout Property

Returns or sets the ODBC query time limit, in seconds. The default value is 45 seconds. Read/write **Long**.

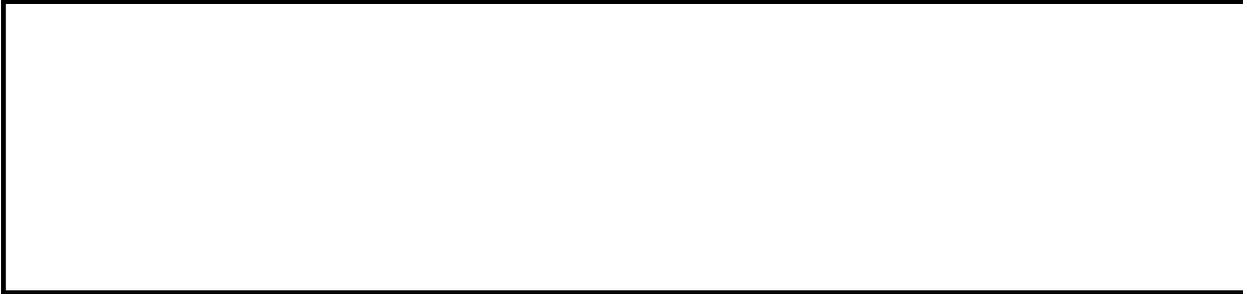
## Remarks

The value 0 (zero) indicates an indefinite time limit.

## Example

This example sets the ODBC query time limit to 15 seconds.

```
Application.ODBCTimeout = 15
```



[Show All](#)

# Offset Property

 [Offset property as it applies to the \*\*Range\*\* object.](#)

Returns a **Range** object that represents a range that's offset from the specified range. Read-only.

*expression*.**Offset**(*RowOffset*, *ColumnOffset*)

*expression* Required. An expression that returns a **Range** object.

**RowOffset** Optional **Variant**. The number of rows (positive, negative, or 0 (zero)) by which the range is to be offset. Positive values are offset downward, and negative values are offset upward. The default value is 0.

**ColumnOffset** Optional **Variant**. The number of columns (positive, negative, or 0 (zero)) by which the range is to be offset. Positive values are offset to the right, and negative values are offset to the left. The default value is 0.

 [Offset property as it applies to the \*\*TickLabels\*\* object.](#)

Returns or sets the distance between the levels of labels, and the distance between the first level and the axis line. The default distance is 100 percent, which represents the default spacing between the axis labels and the axis line. The value can be an integer percentage from 0 through 1000, relative to the axis label's font size. Read/write **Long**.

*expression*.**Offset**

*expression* Required. An expression that returns a **TickLabels** object.

## Example

 [As it applies to the \*\*Range\*\* object.](#)

This example activates the cell three columns to the right of and three rows down from the active cell on Sheet1.

```
Worksheets("Sheet1").Activate  
ActiveCell.Offset(rowOffset:=3, columnOffset:=3).Activate
```

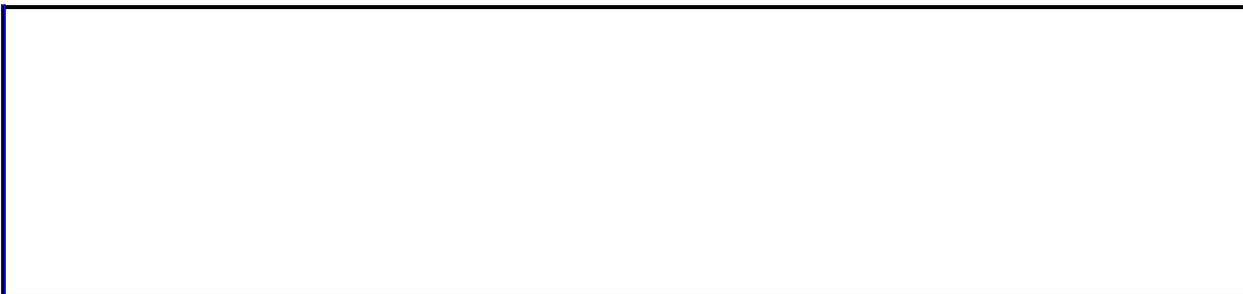
This example assumes that Sheet1 contains a table that has a header row. The example selects the table, without selecting the header row. The active cell must be somewhere in the table before the example is run.

```
Set tbl = ActiveCell.CurrentRegion  
tbl.Offset(1, 0).Resize(tbl.Rows.Count - 1, _  
tbl.Columns.Count).Select
```

 [As it applies to the \*\*TickLabels\*\* object.](#)

This example sets the label spacing of the value axis in Chart1 to twice the current setting, if the offset is less than 500.

```
With Charts("Chart1").Axes(xlValue).TickLabels  
    If .Offset < 500 then  
        .Offset = .Offset * 2  
    End If  
End With
```



# OffsetX Property

Returns or sets the horizontal offset of the shadow from the specified shape, in points. A positive value offsets the shadow to the right of the shape; a negative value offsets it to the left. Read/write **Single**.

*expression*.**OffsetX**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

If you want to nudge a shadow horizontally or vertically from its current position without having to specify an absolute position, use the [IncrementOffsetX](#) method or the [IncrementOffsetY](#) method.

## Example

This example sets the horizontal and vertical offsets for the shadow of shape three on myDocument. The shadow is offset 5 points to the right of the shape and 3 points above it. If the shape doesn't already have a shadow, this example adds one to it.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(3).Shadow
    .Visible = True
    .OffsetX = 5
    .OffsetY = -3
End With
```

# OffsetY Property

Returns or sets the vertical offset of the shadow from the specified shape, in points. A positive value offsets the shadow to the right of the shape; a negative value offsets it to the left. Read/write **Single**.

## Remarks

If you want to nudge a shadow horizontally or vertically from its current position without having to specify an absolute position, use the [IncrementOffsetX](#) method or the [IncrementOffsetY](#) method.

## Example

This example sets the horizontal and vertical offsets for the shadow of shape three on myDocument. The shadow is offset 5 points to the right of the shape and 3 points above it. If the shape doesn't already have a shadow, this example adds one to it.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(3).Shadow
    .Visible = True
    .OffsetX = 5
    .OffsetY = -3
End With
```



[Show All](#)

# OLAP Property

Returns **True** if the PivotTable cache is connected to an [Online Analytical Processing \(OLAP\)](#) server. Read-only **Boolean**.

*expression*.**OLAP**

*expression* Required. An expression that returns a [PivotCache](#) object.

## Example

This example determines if the cache connection is to an OLAP server or not. The example assumes a PivotTable exists on the active worksheet.

```
Sub CheckPivotCache()  
    ' Determine if PivotCache has OLAP connection.  
    If Application.ActiveWorkbook.PivotCaches.Item(1).OLAP = True Th  
        MsgBox "The PivotCache is connected to an OLAP server"  
    Else  
        MsgBox "The PivotCache is not connected to an OLAP server."  
    End If  
End Sub
```



# OLEDBErrors Property

Returns the [OLEDBErrors](#) collection, which represents the error information returned by the most recent OLE DB query. Read-only.

## Example

This example displays the error description and **SqlState** property value for an OLE DB error returned by the most recent OLE DB query.

```
Set objEr = Application.OLEDBErrors.Item(1)
MsgBox "The following error occurred:" & _
    objEr.ErrorString & " : " & objEr.SqlState
```



# OLEFormat Property

Returns an [OLEFormat](#) object that contains OLE object properties. Read-only.

## Example

This example activates an OLE object. If Shapes(1) doesn't represent an embedded OLE object, this example fails..

```
Worksheets(1).Shapes(1).OLEFormat.Activate
```



# OLEType Property

Returns the OLE object type. Can be one of the following **XIOLEType** constants: **xIOLELink** or **xIOLEEmbed**. Returns **xIOLELink** if the object is linked (it exists outside of the file), or returns **xIOLEEmbed** if the object is embedded (it's entirely contained within the file). Read-only **Long**.

## Example

This example creates a list of link types for OLE objects on Sheet1. The list appears on a new worksheet created by the example.

```
Set newSheet = Worksheets.Add
i = 2
newSheet.Range("A1").Value = "Name"
newSheet.Range("B1").Value = "Link Type"
For Each obj In Worksheets("Sheet1").OLEObjects
    newSheet.Cells(i, 1).Value = obj.Name
    If obj.OLEType = xlOLELink Then
        newSheet.Cells(i, 2) = "Linked"
    Else
        newSheet.Cells(i, 2) = "Embedded"
    End If
    i = i + 1
Next
```



# OmittedCells Property

When set to **True** (default), Microsoft Excel identifies, with an AutoCorrect Options button, the selected cells that contain formulas referring to a range that omits adjacent cells that could be included. **False** disables error checking for omitted cells. Read/write **Boolean**.

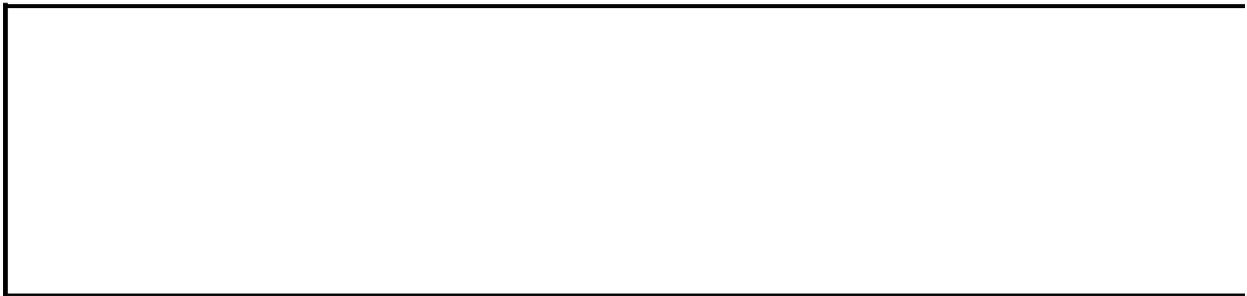
*expression*.**OmittedCells**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In the following example, the **AutoCorrect Options** button appears for cell A4, which contains a formula.

```
Sub CheckOmittedCells()  
  
    Application.ErrorCheckingOptions.OmittedCells = True  
    Range("A1").Value = 1  
    Range("A2").Value = 2  
    Range("A3").Value = 3  
    Range("A4").Formula = "=Sum(A1:A2)"  
  
End Sub
```



# On Property

**True** if the specified filter is on. Read-only **Boolean**.

## Example

The following example sets a variable to the value of the **Criteria1** property of the filter for the first column in the filtered range on the Crew worksheet.

```
With Worksheets("Crew")
    If .AutoFilterMode Then
        With .AutoFilter.Filters(1)
            If .On Then c1 = .Criteria1
        End With
    End If
End With
```



# OnAction Property

Returns or sets the name of a macro that's run when the specified object is clicked. Read/write **String**.

## **Remarks**

Setting this property for a menu item overrides any custom help information set up for the menu item with the information set up for the assigned macro.

## Example

This example causes Microsoft Excel to run the ShapeClick procedure whenever shape one is clicked.

```
Worksheets(1).Shapes(1).OnAction = "ShapeClick"
```



# OnWindow Property

Returns or sets the name of the procedure that's run whenever you activate a window. Read/write **String**.

## Remarks

The procedure specified by this property isn't run when other procedures switch to the window or when a command to switch to a window is received through a DDE channel. Instead, the procedure responds to the user's actions, such as clicking a window with the mouse, clicking **Go To** on the **Edit** menu, and so on.

If a worksheet or macro sheet has an Auto\_Activate or Auto\_Deactivate macro defined for it, those macros will be run after the procedure specified by the **OnWindow** property.

## Example

This example causes the `WindowActivate` procedure to be run whenever window one is activated.

```
ThisWorkbook.Windows(1).OnWindow = "WindowActivate"
```



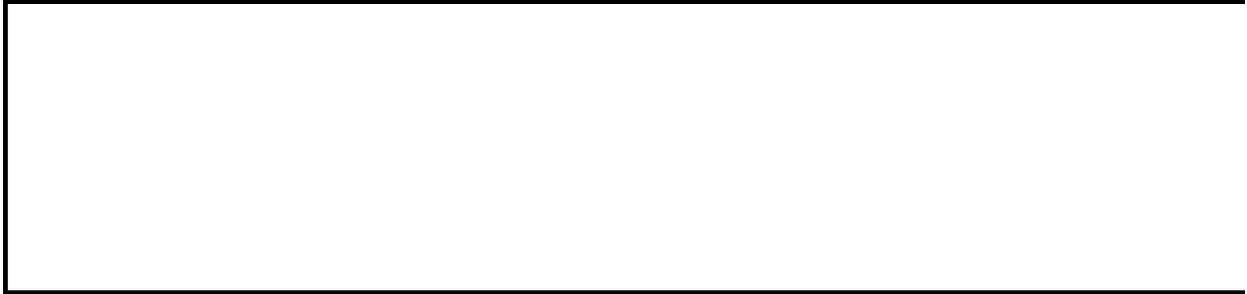
# OperatingSystem Property

Returns the name and version number of the current operating system— for example, "Windows (32-bit) 4.00" or "Macintosh 7.00". Read-only **String**.

## Example

This example displays the name of the operating system.

```
MsgBox "Microsoft Excel is using " & Application.OperatingSystem
```



[Show All](#)

# Operator Property

 [Operator property as it applies to the \*\*Filter\*\* object.](#)

Returns the operator that associates the two criteria applied by the specified filter. Read-only [XIAutoFilterOperator](#).

XIAutoFilterOperator can be one of these XIAutoFilterOperator constants.

**xlAnd**

**xlBottom10Percent**

**xlTop10Items**

**xlBottom10Items**

**xlOr**

**xlTop10Percent**

*expression*.**Operator**

*expression* Required. An expression that returns one of the above objects.

 [Operator property as it applies to the \*\*FormatCondition\*\* and \*\*Validation\*\* objects.](#)

Returns the operator for the conditional format or data validation. Read-only **Long**.

*expression*.**Operator**

*expression* Required. An expression that returns one of the above objects.

## Example

[As it applies to the \*\*FormatCondition\*\* object.](#)

This example changes the formula for conditional format one, for cells E1:E10 if the formula specifies "less than 5."

```
With Worksheets(1).Range("e1:e10").FormatConditions(1)
    If .Operator = xlLess And .Formula1 = "5" Then
        .Modify xlCellValue, xlBetween, "5", "15"
    End If
End With
```



# OptimizeCache Property

**True** if the PivotTable cache is optimized when it's constructed. The default value is **False**. Read/write **Boolean**.

## Remarks

Cache optimization results in additional queries and degrades initial performance of the PivotTable report.

For OLE DB data sources, this property is read-only and always returns **False**.

## Example

This example causes the PivotTable cache for the first PivotTable report on worksheet one to be optimized when it's constructed.

```
Worksheets(1).PivotTables("Pivot1") _  
    .PivotCache.OptimizeCache = True
```



[Show All](#)

# Order Property

[Order property as it applies to the \*\*PageSetup\*\* object.](#)

Returns or sets the order that Microsoft Excel uses to number pages when printing a large worksheet. Read/write **XIOrder**.

XIOrder can be one of these XIOrder constants.

**xlDownThenOver**

**xlOverThenDown**

*expression*.**Order**

*expression* Required. An expression that returns one of the above objects.

[Order property as it applies to the \*\*Trendline\*\* object.](#)

Returns or sets the trendline order (an integer greater than 1) when the trendline type is **xlPolynomial**. Read/write **Long**.

*expression*.**Order**

*expression* Required. An expression that returns one of the above objects.

## Example

This example breaks Sheet1 into pages when the worksheet is printed. Numbering and printing proceed from the first page to the pages to the right, and then move down and continue printing across the sheet.

```
Worksheets("Sheet1").PageSetup.Order = xlOverThenDown
```



# OrganizationName Property

Returns the registered organization name. Read-only **String**.

## Example

This example displays the registered organization name.

```
MsgBox "The registered organization is " & _  
    Application.OrganizationName
```



# OrganizeInFolder Property

**True** if all supporting files, such as background textures and graphics, are organized in a separate folder when you save the specified document as a Web page. **False** if supporting files are saved in the same folder as the Web page. The default value is **True**. Read/write **Boolean**.

## Remarks

The new folder is created in the folder where you have saved the Web page, and is named after the document. If long file names are used, a suffix is added to the folder name. The [FolderSuffix](#) property returns the folder suffix for the language support you have selected or installed, or the default folder suffix.

If you save a document that was previously saved with the **OrganizeInFolder** property set to a different value, Microsoft Excel automatically moves the supporting files into or out of the folder, as appropriate.

If you don't use long file names (that is, if the [UseLongFileNames](#) property is set to **False**), Microsoft Excel automatically saves any supporting files in a separate folder. The files cannot be saved in the same folder as the Web page.

## Example

This example specifies that all supporting files are saved in the same folder when the document is saved as a Web page.

```
Application.DefaultWebOptions.OrganizeInFolder = False
```



[Show All](#)

# Orientation Property

 [Orientation property as it applies to the \*\*TextFrame\*\* object.](#)

The text frame orientation. Can be an integer value from – 90 to 90 degrees or one of the **MsoTextOrientation** constants. Read/write [MsoTextOrientation](#).

MsoTextOrientation can be one of these MsoTextOrientation constants.

**msoTextOrientationDownward**

**msoTextOrientationHorizontal**

**msoTextOrientationHorizontalRotatedFarEast**

**msoTextOrientationMixed**

**msoTextOrientationUpward**

**msoTextOrientationVertical**

**msoTextOrientationVerticalFarEast**

*expression*.**Orientation**

*expression* Required. An expression that returns a **TextFrame** object.

 [Orientation property as it applies to the \*\*Style\*\* object.](#)

The text orientation. Can be an integer value from – 90 to 90 degrees or one of the **XlOrientation** constants. Read/write [XlOrientation](#).

XlOrientation can be one of these XlOrientation constants.

**xlDownward**

**xlUpward**

**xlHorizontal**

**xlVertical**

*expression*.**Orientation**

*expression* Required. An expression that returns a **Style** object.

 [Orientation property as it applies to the \*\*PageSetup\*\* object.](#)

Portrait or landscape printing mode. Read/write [XlPageOrientation](#).

XlPageOrientation can be one of these XlPageOrientation constants.

**xlPortrait**

**xlLandscape**

*expression*.**Orientation**

*expression* Required. An expression that returns a **PageSetup** object.

 [Orientation property as it applies to the \*\*CubeField\*\* and \*\*PivotField\*\* objects.](#)

The location of the field in the specified PivotTable report. Read/write [XlPivotFieldOrientation](#).

XlPivotFieldOrientation can be one of these XlPivotFieldOrientation constants.

**xlColumnField**

**xldataField**

**xlHidden**

**xlPageField**

**xlRowField**

*expression*.**Orientation**

*expression* Required. An expression that returns one of the above objects.

 [Orientation property as it applies to the \*\*TickLabels\*\* object.](#)

The text orientation. Can be an integer value from – 90 to 90 degrees or one of the **XlTickLabelOrientation** constants. Read/write [XlTickLabelOrientation](#).

XlTickLabelOrientation can be one of these XlTickLabelOrientation constants.

**xlTickLabelOrientationAutomatic**

**xlTickLabelOrientationHorizontal**

**xlTickLabelOrientationVertical**  
**xlTickLabelOrientationDownward**  
**xlTickLabelOrientationUpward**

*expression*.**Orientation**

*expression* Required. An expression that returns a **TickLabels** object.

[Orientation property as it applies to the \*\*AxisTitle\*\*, \*\*CellFormat\*\*, \*\*ChartTitle\*\*, \*\*DataLabel\*\*, \*\*DataLabels\*\*, \*\*DisplayUnitLabel\*\*, and \*\*Range\*\* objects.](#)

The text orientation. Can be an integer value from – 90 to 90 degrees. Read/write **Variant**.

*expression*.**Orientation**

*expression* Required. An expression that returns one of the above objects.

## Remarks

For [OLAP](#) data sources, setting this property for one field in a hierarchy sets the orientation for the other fields in the same hierarchy. Dimension fields can only be oriented in the row, column, and page field areas of the PivotTable report. Measure fields can only be oriented in the data area. Setting a hierarchy or data field to **xlHidden** removes the hierarchy or field from the PivotTable report.

## Example

[As it applies to the \*\*PivotField\*\* object.](#)

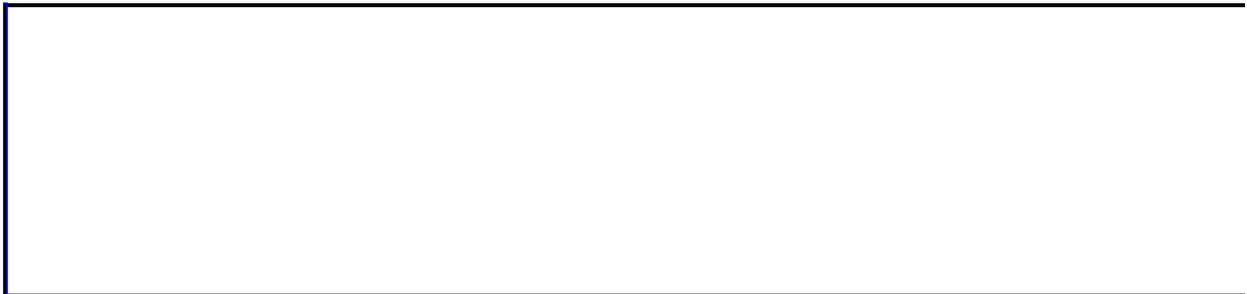
This example displays the orientation for the ORDER\_DATE field.

```
Set pvtTable = Worksheets("Sheet1").Range("A3").PivotTable
Set pvtField = pvtTable.PivotFields("ORDER_DATE")
Select Case pvtField.Orientation
    Case xlHidden
        MsgBox "Hidden field"
    Case xlRowField
        MsgBox "Row field"
    Case xlColumnField
        MsgBox "Column field"
    Case xlPageField
        MsgBox "Page field"
    Case xlDataField
        MsgBox "Data field"
End Select
```

[As it applies to the \*\*PageSetup\*\* object.](#)

This example sets Sheet1 to be printed in landscape orientation.

```
Worksheets("Sheet1").PageSetup.Orientation = xlLandscape
```



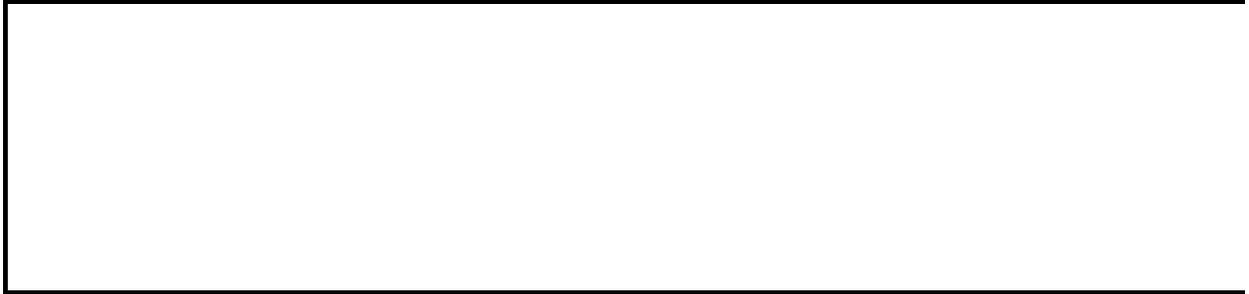
# Outline Property

Returns an [Outline](#) object that represents the outline for the specified worksheet.  
Read-only.

## Example

This example sets the outline on Sheet1 to use automatic styles.

```
worksheets("Sheet1").Outline.AutomaticStyles = True
```



# OutlineFont Property

**True** if the font is an outline font. Read/write **Boolean**.

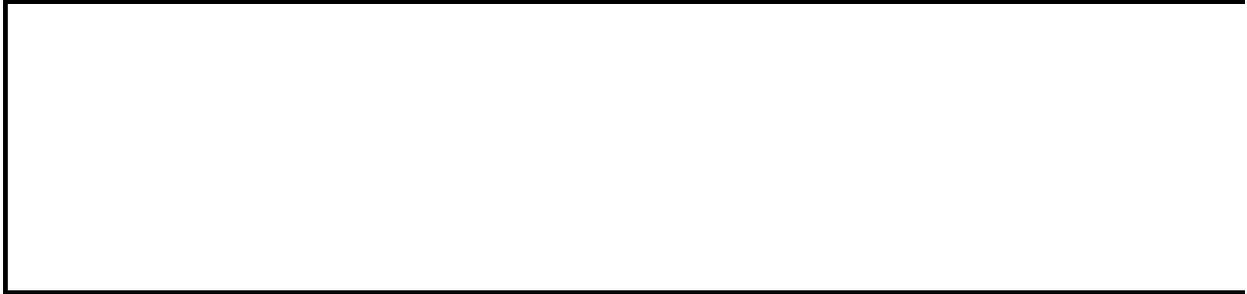
## Remarks

This property has no effect in Windows, but its value is retained (it can be set and returned).

## Example

This example sets the font for cell A1 on Sheet1 to an outline font.

```
Worksheets("Sheet1").Range("A1").Font.OutlineFont = True
```



# OutlineLevel Property

Returns or sets the current outline level of the specified row or column.  
Read/write **Variant**.

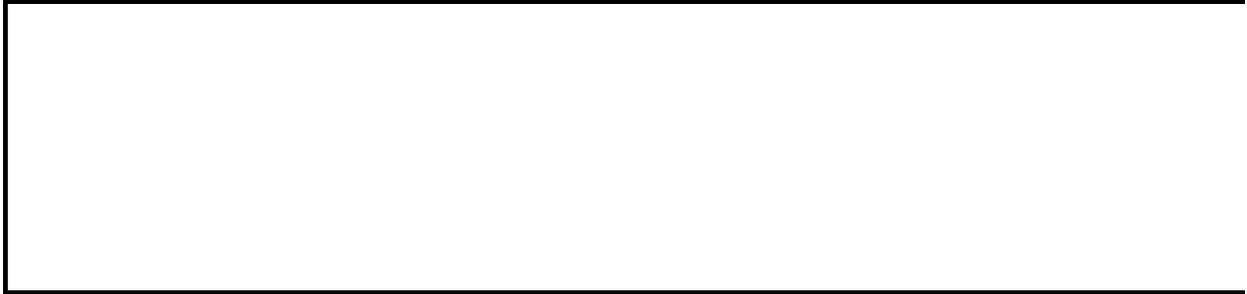
## **Remarks**

Level one is the outermost summary level.

## Example

This example sets the outline level for row two on Sheet1.

```
Worksheets("Sheet1").Rows(2).OutlineLevel = 1
```



# Overlap Property

Specifies how bars and columns are positioned. Can be a value between – 100 and 100. Applies only to 2-D bar and 2-D column charts. Read/write **Long**.

## Remarks

If this property is set to  $-100$ , bars are positioned so that there's one bar width between them. If the overlap is 0 (zero), there's no space between bars (one bar starts immediately after the preceding bar). If the overlap is 100, bars are positioned on top of each other.

## Example

This example sets the overlap for chart group one to – 50. The example should be run on a 2-D column chart that has two or more series.

```
Charts("Chart1").ChartGroups(1).Overlap = -50
```



# PageBreak Property

Returns or sets the location of a page break. Can be one of the following **XlPageBreak** constants: **xlPageBreakAutomatic**, **xlPageBreakManual**, or **xlPageBreakNone**. Read/write **Long**.

## Remarks

This property can return the location of either automatic or manual page breaks, but it can only set the location of manual breaks (it can only be set to **xlPageBreakManual** or **xlPageBreakNone**).

To remove all manual page breaks on a worksheet, set `Cells.PageBreak` to **xlPageBreakNone**.

## Example

This example sets a manual page break above row 25 on Sheet1.

```
Worksheets("Sheet1").Rows(25).PageBreak = xlPageBreakManual
```

This example sets a manual page break to the left of column J on Sheet1.

```
Worksheets("Sheet1").Columns("J").PageBreak = xlPageBreakManual
```

This example deletes the two page breaks that were set in the preceding examples.

```
Worksheets("Sheet1").Rows(25).PageBreak = xlPageBreakNone  
Worksheets("Sheet1").Columns("J").PageBreak = xlNone
```



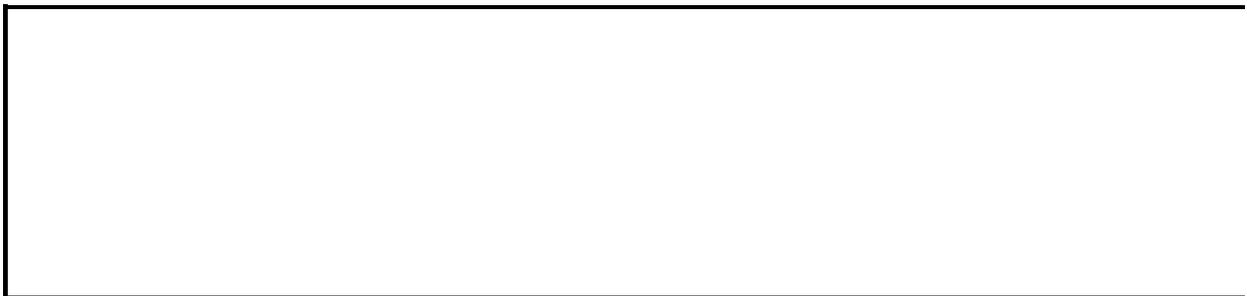
# PageFieldOrder Property

Returns or sets the order in which page fields are added to the PivotTable report's layout. Can be one of the following **XIOrder** constants: **xlDownThenOver** or **xlOverThenDown**. The default constant is **xlDownThenOver**. Read/write **Long**.

## Example

This example causes the PivotTable report to draw three page fields in a row before starting a new row.

```
With Worksheets(1).PivotTables("Pivot1")  
    .PageFieldOrder = xlOverThenDown  
    .PageFieldWrapCount = 3  
End With
```



# PageFields Property

Returns an object that represents either a single PivotTable field (a [PivotField](#) object) or a collection of all the fields (a [PivotFields](#) object) that are currently showing as page fields. Read-only.

*expression*.**PageFields**(*Index*)

*expression* Required. An expression that returns a **PivotTable** object.

**Index** Optional **Variant**. The name or number of the field to be returned (can be an array to specify more than one field).

## Remarks

A hierarchy can contain only one page field.

For a PivotTable report based on a PivotTable cache, the collection of PivotTable fields that's returned reflects what's currently in the cache.

## Example

This example adds the page field names to a list on a new worksheet.

```
Set nwSheet = Worksheets.Add
nwSheet.Activate
Set pvtTable = Worksheets("Sheet2").Range("A1").PivotTable
rw = 0
For Each pvtField In pvtTable.PageFields
    rw = rw + 1
    nwSheet.Cells(rw, 1).Value = pvtField.Name
Next pvtField
```



# PageFieldStyle Property

Returns or sets the style used in the bound page field area. The default value is a null string (no style is applied by default). Read/write **String**.

## **Remarks**

This style is used as the default style for the background area, and it's applied before any user formatting. Cells vacated when a field is pivoted from the page field area to another location retain this style.

## Example

This example sets the page field area of the first PivotTable report on worksheet one to the PurpleAndGold style.

```
Worksheets(1).PivotTables("Pivot1").  
    .PageFieldStyle = "PurpleAndGold"
```



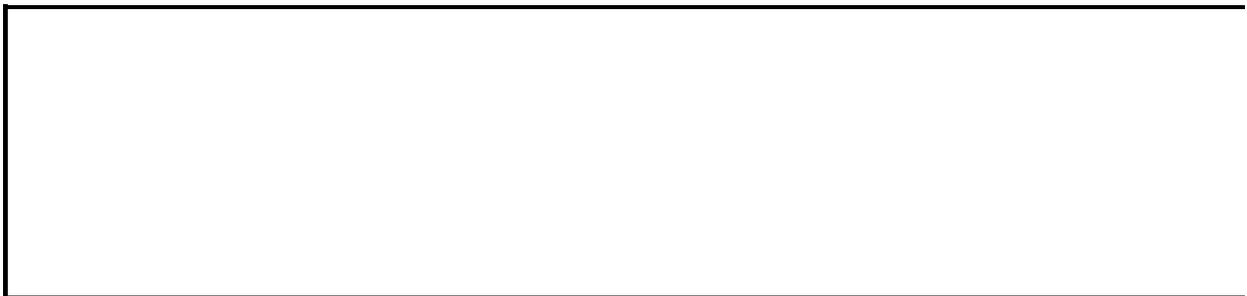
# PageFieldWrapCount Property

Returns or sets the number of page fields in each column or row in the PivotTable report. Read/write **Long**.

## Example

This example causes the PivotTable report to draw three page fields in a row before starting a new row.

```
With Worksheets(1).PivotTables("Pivot1")  
    .PageFieldOrder = xlOverThenDown  
    .PageFieldWrapCount = 3  
End With
```



# PageRange Property

Returns a [Range](#) object that represents the range that contains the page area in the PivotTable report. Read-only.

## Example

This example selects the page headers in the PivotTable report.

```
Worksheets("Sheet1").Activate  
Range("A3").Select  
ActiveCell.PivotTable.PageRange.Select
```



# PageRangeCells Property

Returns a **Range** object that represents only the cells in the specified PivotTable report that contain the page fields and item drop-down lists.

## Example

This example selects only the cells in the PivotTable report that contain page fields and item drop-down lists.

```
Worksheets(1).PivotTables(1).PageRangeCells.Select
```



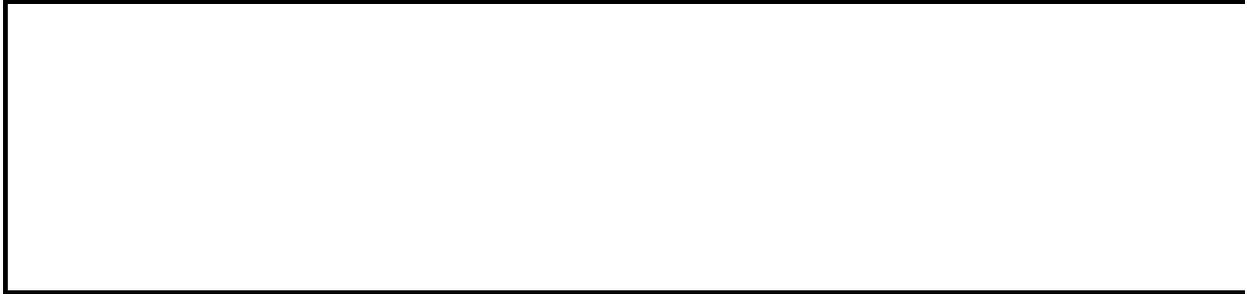
# PageSetup Property

Returns a [PageSetup](#) object that contains all the page setup settings for the specified object. Read-only.

## Example

This example sets the center header text for Chart1.

```
Charts("Chart1").PageSetup.CenterHeader = "December Sales"
```



# Panes Property

Returns a [Panels](#) collection that represents all the panes in the specified window. Read-only.

For information about returning a single member of a collection, see [Returning an Object from a Collection](#).

## Remarks

This property is available for a window only if the window's **Split** property can be set to **True**.

## Example

This example displays the number of panes in the active window in Book1.xls.

```
Workbooks("BOOK1.XLS").Worksheets("Sheet1").Activate  
MsgBox "There are " & ActiveWindow.Panes.Count & _  
    " panes in the active window"
```

This example activates the pane in the upper-left corner of the active window in Book1.xls.

```
Workbooks("BOOK1.XLS").Worksheets("Sheet1").Activate  
ActiveWindow.Panes(1).Activate
```



[Show All](#)

# PaperSize Property

Returns or sets the size of the paper. Read/write [XIPaperSize](#).

XIPaperSize can be one of these XIPaperSize constants.

**xIPaper11x17.** 11 in. x 17 in.

**xIPaperA4.** A4 (210 mm x 297 mm)

**xIPaperA5.** A5 (148 mm x 210 mm)

**xIPaperB5.** A5 (148 mm x 210 mm)

**xIPaperDsheet.** D size sheet

**xIPaperEnvelope11.** Envelope #11 (4-1/2 in. x 10-3/8 in.)

**xIPaperEnvelope14.** Envelope #14 (5 in. x 11-1/2 in.)

**xIPaperEnvelopeB4.** Envelope B4 (250 mm x 353 mm)

**xIPaperEnvelopeB6.** Envelope B6 (176 mm x 125 mm)

**xIPaperEnvelopeC4.** Envelope C4 (229 mm x 324 mm)

**xIPaperEnvelopeC6.** Envelope C6 (114 mm x 162 mm)

**xIPaperEnvelopeDL.** Envelope DL (110 mm x 220 mm)

**xIPaperEnvelopeMonarch.** Envelope Monarch (3-7/8 in. x 7-1/2 in.)

**xIPaperEsheet.** E size sheet

**xIPaperFanfoldLegalGerman.** German Legal Fanfold (8-1/2 in. x 13 in.)

**xIPaperFanfoldUS.** U.S. Standard Fanfold (14-7/8 in. x 11 in.)

**xIPaperLedger.** Ledger (17 in. x 11 in.)

**xIPaperLetter.** Letter (8-1/2 in. x 11 in.)

**xIPaperNote.** Note (8-1/2 in. x 11 in.)

**xIPaperStatement.** Statement (5-1/2 in. x 8-1/2 in.)

**xIPaperUser.** User-defined

**xIPaper10x14.** 10 in. x 14 in.

**xIPaperA3.** A3 (297 mm x 420 mm)

**xIPaperA4Small.** A4 Small (210 mm x 297 mm)

**xIPaperB4.** B4 (250 mm x 354 mm)

**xIPaperCsheet.** C size sheet

**xIPaperEnvelope10.** Envelope #10 (4-1/8 in. x 9-1/2 in.)

**xlPaperEnvelope12.** Envelope #12 (4-1/2 in. x 11 in.)  
**xlPaperEnvelope9.** Envelope #9 (3-7/8 in. x 8-7/8 in.)  
**xlPaperEnvelopeB5.** Envelope B5 (176 mm x 250 mm)  
**xlPaperEnvelopeC3.** Envelope C3 (324 mm x 458 mm)  
**xlPaperEnvelopeC5.** Envelope C5 (162 mm x 229 mm)  
**xlPaperEnvelopeC65.** Envelope C65 (114 mm x 229 mm)  
**xlPaperEnvelopeItaly.** Envelope (110 mm x 230 mm)  
**xlPaperEnvelopePersonal.** Envelope (3-5/8 in. x 6-1/2 in.)  
**xlPaperExecutive.** Executive (7-1/2 in. x 10-1/2 in.)  
**xlPaperFanfoldStdGerman.** German Legal Fanfold (8-1/2 in. x 13 in.)  
**xlPaperFolio.** Folio (8-1/2 in. x 13 in.)  
**xlPaperLegal.** Legal (8-1/2 in. x 14 in.)  
**xlPaperLetterSmall.** Letter Small (8-1/2 in. x 11 in.)  
**xlPaperQuarto.** Quarto (215 mm x 275 mm)  
**xlPaperTabloid.** Tabloid (11 in. x 17 in.)

**Note** Some printers may not support all of these paper sizes.

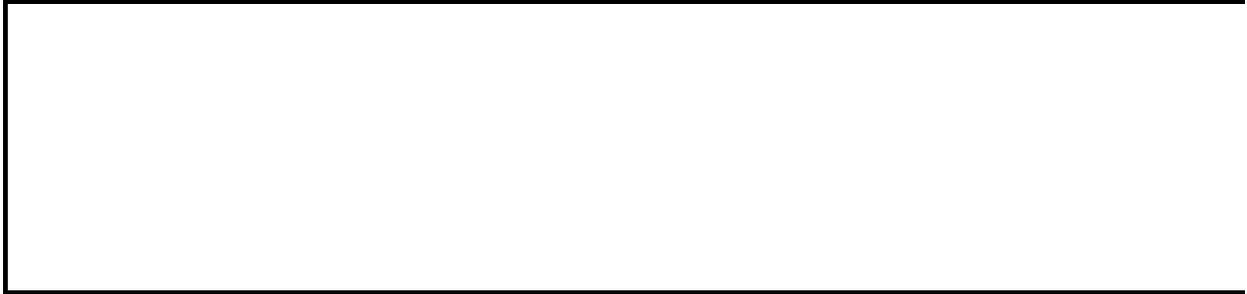
### *expression.***PaperSize**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example sets the paper size to legal for Sheet1.

```
Worksheets("Sheet1").PageSetup.PaperSize = xlPaperLegal
```



# Parameters Property

Returns a [Parameters](#) collection that represents the query table parameters.  
Read-only.

For more information about returning a single object from a collection, see [Returning an Object from a Collection](#).

## Example

This example returns the **Parameters** collection from an existing parameter query. If the first parameter uses the character data type, the user is instructed to enter characters only in the prompt dialog box.

```
With Sheets("sheet1").QueryTables(1).Parameters(1)
    If .DataType = xlParamTypeVarChar Then
        .SetParam xlPrompt, "Enter a character only"
    End If
End With
```



# Parent Property

Returns the parent object for the specified object. Read-only.

*expression*.**Parent**

*expression* Required. An expression that returns one of the objects in the Applies To list.

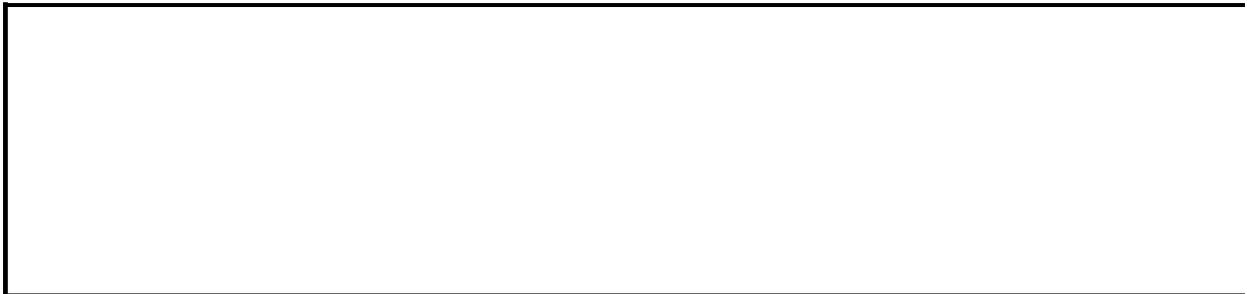
## Example

This example displays the name of the chart that contains myAxis.

```
Sub DisplayParentName()  
    Set myAxis = Charts(1).Axes(xlValue)  
    MsgBox myAxis.Parent.Name  
End Sub
```

This example displays the name of the **ListColumn** object containing the **ListDataFormat** object for the third column of a list in Sheet1 of the active workbook.

```
Sub PrintParentName()  
    Dim wrksht As Worksheet  
    Dim objListCol As ListColumn  
  
    Set wrksht = ActiveWorkbook.Worksheets("Sheet1")  
    Set objListCol = wrksht.ListObjects(1).ListColumns(3)  
  
    Debug.Print objListCol.ListDataFormat.Parent  
End Sub
```



[Show All](#)

# ParentField Property

Returns a [PivotField](#) object that represents the PivotTable field that's the group parent of the specified object. The field must be grouped and must have a parent field. Read-only.

## Remarks

This property isn't available for [OLAP](#) data sources.

## Example

This example displays the name of the field that's the group parent of the field that contains the active cell.

```
Worksheets("Sheet1").Activate  
MsgBox "The active field is a child of the field " & _  
    ActiveCell.PivotField.ParentField.Name
```



# ParentGroup Property

Returns a [Shape](#) object that represents the common parent shape of a child shape or a range of child shapes.

*expression*.**ParentGroup**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel adds two shapes to the active worksheet and then removes both shapes by deleting the parent shape of the group.

```
Sub ParentGroup()  
    Dim pgShape As Shape  
  
    With ActiveSheet.Shapes  
        .AddShape Type:=1, Left:=10, Top:=10, _  
            Width:=100, Height:=100  
        .AddShape Type:=2, Left:=110, Top:=120, _  
            Width:=100, Height:=100  
        .Range(Array(1, 2)).Group  
    End With  
  
    ' Using the child shape in the group get the Parent shape.  
    Set pgShape = ActiveSheet.Shapes(1).GroupItems(1).ParentGroup  
  
    MsgBox "The two shapes will now be deleted."  
  
    ' Delete the parent shape.  
    pgShape.Delete  
  
End Sub
```



[Show All](#)

# ParentItem Property

Returns a [PivotItem](#) object that represents the parent PivotTable item in the parent [PivotField](#) object (the field must be grouped so that it has a parent).  
Read-only.

## Remarks

This property isn't available for [OLAP](#) data sources.

## Example

This example displays the name of the parent item for the item that contains the active cell.

```
Worksheets("Sheet1").Activate  
MsgBox "This item is a subitem of " & _  
    ActiveCell.PivotItem.ParentItem.Name
```



[Show All](#)

# ParentItems Property

Returns an object that represents either a single PivotTable item (a [PivotItem](#) object) or a collection of all the items (a [PivotItems](#) object) that are group parents in the specified field. The specified field must be a group parent of another field. Read-only.

*expression*.**ParentItems**(*Index*)

*expression* Required. An expression that returns a **PivotField** object.

**Index** Optional **Variant**. The number or name of the item to be returned (can be an array to specify more than one item).

## Remarks

This property isn't available for [OLAP](#) data sources.

## Example

This example creates a list containing the names of all the items that are group parents in the field named "product".

```
Set nwSheet = Worksheets.Add
nwSheet.Activate
Set pvtTable = Worksheets("Sheet2").Range("A1").PivotTable
rw = 0
For Each pvtItem In pvtTable.PivotFields("product").ParentItems
    rw = rw + 1
    nwSheet.Cells(rw, 1).Value = pvtItem.Name
Next pvtItem
```



[Show All](#)

# ParentShowDetail Property

**True** if the specified item is showing because one of its parents is showing detail. **False** if the specified item isn't showing because one of its parents is hiding detail. This property is available only if the item is grouped. Read-only **Boolean**.

## Remarks

This property isn't available for [OLAP](#) data sources.

## Example

This example displays a message if the item that contains the active cell is visible because its parent item is showing detail.

```
Worksheets("Sheet1").Activate  
Set pvtItem = ActiveCell.PivotItem  
If pvtItem.ParentShowDetail = True Then  
    MsgBox "Parent item is showing detail"  
End If
```



# Password Property

Returns or sets the password that must be supplied to open the specified workbook. Read/write **String**.

Use strong passwords that combine upper- and lowercase letters, numbers, and symbols. Weak passwords don't mix these elements. Strong password: Y6dh!et5. Weak password: House27. Use a strong password that you can remember so that you don't have to write it down.

*expression*.**Password**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel opens a workbook named Password.xls, sets a password for it, and then closes the workbook. This example assumes a file named "Password.xls" exists on the C:\ drive.

```
Sub UsePassword()  
    Dim wkbOne As Workbook  
    Set wkbOne = Application.Workbooks.Open("C:\Password.xls")  
    wkbOne.Password = InputBox ("Enter Password")  
    wkbOne.Close  
End Sub
```

**Note** The **Password** property is readable and returns ">>\*\*\*".



# PasswordEncryptionAlgorithm Property

Returns a **String** indicating the algorithm Microsoft Excel uses to encrypt passwords for the specified workbook. Read-only.

*expression*.**PasswordEncryptionAlgorithm**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Use the [SetPasswordEncryptionOptions](#) method to specify whether Excel encrypts file properties for password-protected workbooks.

## Example

This example sets the password encryption options for the active workbook.

```
Sub SetPasswordOptions()
```

```
    ActiveWorkbook.SetPasswordEncryptionOptions _  
        PasswordEncryptionProvider:="Microsoft RSA SChannel Cryptogr  
        PasswordEncryptionAlgorithm:="RC4", _  
        PasswordEncryptionKeyLength:=56, _  
        PasswordEncryptionFileProperties:=True
```

```
End Sub
```



# PasswordEncryptionFileProperties Property

**True** if Microsoft Excel encrypts file properties for the specified password-protected workbook. Read-only **Boolean**.

*expression*.**PasswordEncryptionFileProperties**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Use the [SetPasswordEncryptionOptions](#) method to specify whether Excel encrypts file properties for the specified password-protected workbook.

## Example

This example sets the password encryption options if the file properties are not encrypted for password-protected workbooks.

```
Sub SetPasswordOptions()  
    With ActiveWorkbook  
        If .PasswordEncryptionFileProperties = False Then  
            .SetPasswordEncryptionOptions _  
                PasswordEncryptionProvider:="Microsoft RSA SChannel  
                PasswordEncryptionAlgorithm:="RC4", _  
                PasswordEncryptionKeyLength:=56, _  
                PasswordEncryptionFileProperties:=True  
        End If  
    End With  
End Sub
```



# PasswordEncryptionKeyLength Property

Returns a **Long** indicating the key length of the algorithm Microsoft Excel uses when encrypting passwords for the specified workbook. Read-only.

*expression*.**PasswordEncryptionKeyLength**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Use the [SetPasswordEncryptionOptions](#) method to specify whether Excel encrypts file properties for the specified password-protected workbook.

## Example

This example sets the password encryption options for the specified workbook, if the password encryption key length is less than 56.

```
Sub SetPasswordOptions()  
    With ActiveWorkbook  
        If .PasswordEncryptionKeyLength < 56 Then  
            .SetPasswordEncryptionOptions _  
                PasswordEncryptionProvider:="Microsoft RSA SChannel  
                PasswordEncryptionAlgorithm:="RC4", _  
                PasswordEncryptionKeyLength:=56, _  
                PasswordEncryptionFileProperties:=True  
        End If  
    End With  
End Sub
```



# PasswordEncryptionProvider Property

Returns a **String** specifying the name of the algorithm encryption provider that Microsoft Excel uses when encrypting passwords for the specified workbook.  
Read-only.

*expression*.**PasswordEncryptionProvider**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example sets the password encryption options for the specified workbook, if the file properties are not encrypted for password-protected workbooks.

```
Sub SetPasswordOptions()  
    With ActiveWorkbook  
        If .PasswordEncryptionProvider <> "Microsoft RSA SChannel Cr  
            .SetPasswordEncryptionOptions _  
                PasswordEncryptionProvider:="Microsoft RSA SChannel  
                PasswordEncryptionAlgorithm:="RC4", _  
                PasswordEncryptionKeyLength:=56, _  
                PasswordEncryptionFileProperties:=True  
        End If  
    End With  
End Sub
```



[Show All](#)

# Path Property

[Path property as it applies to the \*\*AutoRecover\*\* object.](#)

Sets or returns the complete path to where Microsoft Excel will store the AutoRecover temporary files. Read/write **String**.

*expression*.**Path**

*expression* Required. An expression that returns an [AutoRecover](#) object.

[Path property as it applies to the \*\*AddIn\*\*, \*\*Application\*\*, \*\*RecentFile\*\*, and \*\*Workbook\*\* objects.](#)

Returns the complete path to the application, excluding the final separator and name of the application. Read-only **String**.

*expression*.**Path**

*expression* Required. An expression that returns one of the above objects.

## Example

[As it applies to the \*\*AutoRecover\*\* object.](#)

This example sets the path of the AutoRecover file to drive C.

```
Sub SetPath()  
    Application.AutoRecover.Path = "C:\"  
End Sub
```

[As it applies to the \*\*AddIn, Application, RecentFile, and Workbook\*\* objects.](#)

This example displays the complete path to Microsoft Excel.

```
Sub TotalPath()  
    MsgBox "The path is " & Application.Path  
End Sub
```



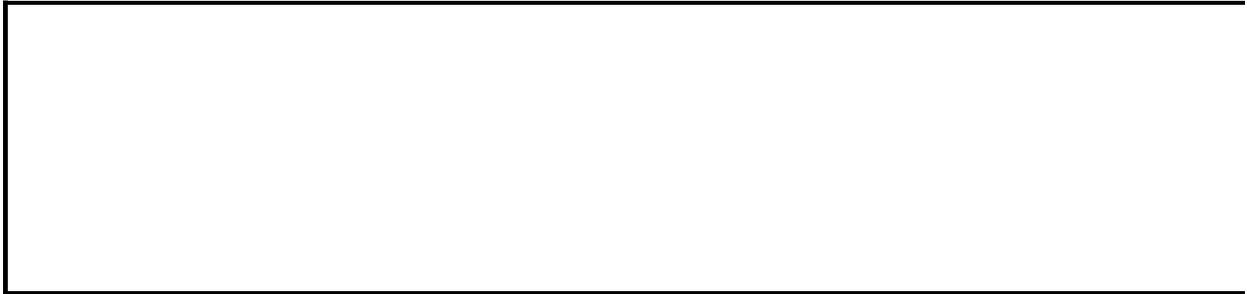
# PathSeparator Property

Returns the path separator character ("\"). Read-only **String**.

## Example

This example displays the current path separator.

```
MsgBox "The path separator character is " & _  
Application.PathSeparator
```



[Show All](#)

# Pattern Property

 [Pattern property as it applies to the \*\*LineFormat\*\* object.](#)

Returns or sets the fill pattern. Read/write [MsoPatternType](#).

MsoPatternType can be one of these MsoPatternType constants.

**msoPattern10Percent**

**msoPattern20Percent**

**msoPattern25Percent**

**msoPattern30Percent**

**msoPattern40Percent**

**msoPattern50Percent**

**msoPattern5Percent**

**msoPattern60Percent**

**msoPattern70Percent**

**msoPattern75Percent**

**msoPattern80Percent**

**msoPattern90Percent**

**msoPatternDarkDownwardDiagonal**

**msoPatternDarkHorizontal**

**msoPatternDarkUpwardDiagonal**

**msoPatternDarkVertical**

**msoPatternDashedDownwardDiagonal**

**msoPatternDashedHorizontal**

**msoPatternDashedUpwardDiagonal**

**msoPatternDashedVertical**

**msoPatternDiagonalBrick**

**msoPatternDivot**

**msoPatternDottedDiamond**

**msoPatternDottedGrid**

**msoPatternHorizontalBrick**

**msoPatternLargeCheckerBoard**  
**msoPatternLargeConfetti**  
**msoPatternLargeGrid**  
**msoPatternLightDownwardDiagonal**  
**msoPatternLightHorizontal**  
**msoPatternLightUpwardDiagonal**  
**msoPatternLightVertical**  
**msoPatternMixed**  
**msoPatternNarrowHorizontal**  
**msoPatternNarrowVertical**  
**msoPatternOutlinedDiamond**  
**msoPatternPlaid**  
**msoPatternShingle**  
**msoPatternSmallCheckerBoard**  
**msoPatternSmallConfetti**  
**msoPatternSmallGrid**  
**msoPatternSolidDiamond**  
**msoPatternSphere**  
**msoPatternTrellis**  
**msoPatternWave**  
**msoPatternWeave**  
**msoPatternWideDownwardDiagonal**  
**msoPatternWideUpwardDiagonal**  
**msoPatternZigZag**

*expression*.**Pattern**

*expression* Required. An expression that returns one of the above objects.

 [Pattern property as it applies to the \*\*ChartFillFormat\*\* and \*\*FillFormat\*\* objects.](#)

Returns or sets the fill pattern. Read-only [MsoPatternType](#).

MsoPatternType can be one of these MsoPatternType constants.

**msoPattern10Percent**  
**msoPattern20Percent**  
**msoPattern25Percent**  
**msoPattern30Percent**  
**msoPattern40Percent**  
**msoPattern50Percent**  
**msoPattern5Percent**  
**msoPattern60Percent**  
**msoPattern70Percent**  
**msoPattern75Percent**  
**msoPattern80Percent**  
**msoPattern90Percent**  
**msoPatternDarkDownwardDiagonal**  
**msoPatternDarkHorizontal**  
**msoPatternDarkUpwardDiagonal**  
**msoPatternDarkVertical**  
**msoPatternDashedDownwardDiagonal**  
**msoPatternDashedHorizontal**  
**msoPatternDashedUpwardDiagonal**  
**msoPatternDashedVertical**  
**msoPatternDiagonalBrick**  
**msoPatternDivot**  
**msoPatternDottedDiamond**  
**msoPatternDottedGrid**  
**msoPatternHorizontalBrick**  
**msoPatternLargeCheckerBoard**  
**msoPatternLargeConfetti**  
**msoPatternLargeGrid**  
**msoPatternLightDownwardDiagonal**  
**msoPatternLightHorizontal**  
**msoPatternLightUpwardDiagonal**  
**msoPatternLightVertical**  
**msoPatternMixed**

**msoPatternNarrowHorizontal**  
**msoPatternNarrowVertical**  
**msoPatternOutlinedDiamond**  
**msoPatternPlaid**  
**msoPatternShingle**  
**msoPatternSmallCheckerBoard**  
**msoPatternSmallConfetti**  
**msoPatternSmallGrid**  
**msoPatternSolidDiamond**  
**msoPatternSphere**  
**msoPatternTrellis**  
**msoPatternWave**  
**msoPatternWeave**  
**msoPatternWideDownwardDiagonal**  
**msoPatternWideUpwardDiagonal**  
**msoPatternZigZag**

*expression*.**Pattern**

*expression* Required. An expression that returns one of the above objects.

 [Pattern property as it applies to the \*\*Interior\*\* object.](#)

Returns or sets the interior pattern. Read/write **Variant**.

*expression*.**Pattern**

*expression* Required. An expression that returns one of the above objects.

## Example

This example adds a crisscross pattern to the interior of cell A1 on Sheet1.

```
Worksheets("Sheet1").Range("A1").  
    Interior.Pattern = xlPatternCrissCross
```



# PatternColor Property

Returns or sets the color of the interior pattern as an RGB value. Read/write **Variant**.

## Example

This example sets the color of the interior pattern for rectangle one on Sheet1.

```
With Worksheets("Sheet1").Rectangles(1).Interior  
    .Pattern = xlGrid  
    .PatternColor = RGB(255,0,0)  
End With
```



# PatternColorIndex Property

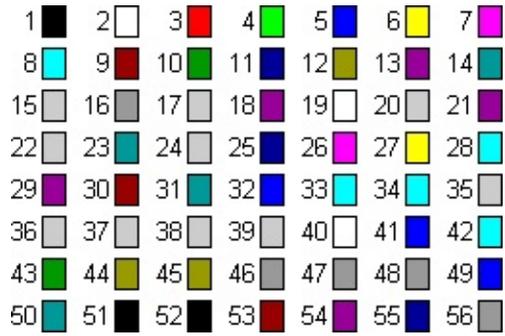
Returns or sets the color of the interior pattern as an index into the current color palette, or as one of the following **XIColorIndex** constants:  
**xIColorIndexAutomatic** or **xIColorIndexNone**. Read/write **Long**.

## Remarks

Set this property to **xlColorIndexAutomatic** to specify the automatic pattern for cells or the automatic fill style for drawing objects. Set this property to **xlColorIndexNone** to specify that you don't want a pattern (this is the same as setting the **Pattern** property of the **Interior** object to **xlPatternNone**).

# Remarks

The following illustration shows the color-index values in the default color palette.



## Example

This example sets the color of the interior pattern for rectangle one on Sheet1.

```
With Worksheets("Sheet1").Rectangles(1).Interior  
    .Pattern = xlChecker  
    .PatternColorIndex = 5  
End With
```



# Period Property

Returns or sets the period for the moving-average trendline. Read/write **Long**.

## Example

This example sets the period for the moving-average trendline on Chart1. The example should be run on a 2-D column chart with a single series that contains 10 data points and a moving-average trendline.

```
With Charts("Chart1").SeriesCollection(1).Trendlines(1)
    If .Type = xlMovingAvg Then .Period = 5
End With
```



# Permission Property

Returns a **Permission** object that represents the permission settings in the specified workbook.

*expression*.**Permission**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

The following example returns the permission settings for the active workbook.

```
Dim objPermission As Permission
```

```
Set objPermission = ActiveWorkbook.Permission
```



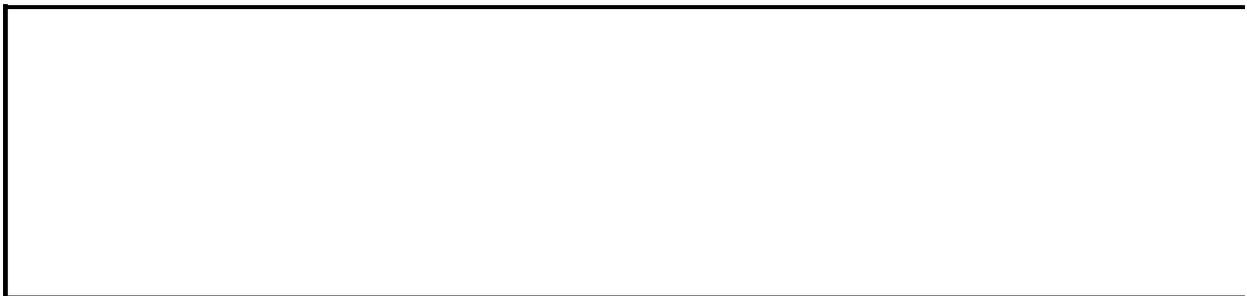
# PersonalViewListSettings Property

**True** if filter and sort settings for lists are included in the user's personal view of the shared workbook. Read/write **Boolean**.

## Example

This example removes print settings and filter and sort settings from the user's personal view of workbook two.

```
With Workbooks(2)  
    .PersonalViewListSettings = False  
    .PersonalViewPrintSettings = False  
End With
```



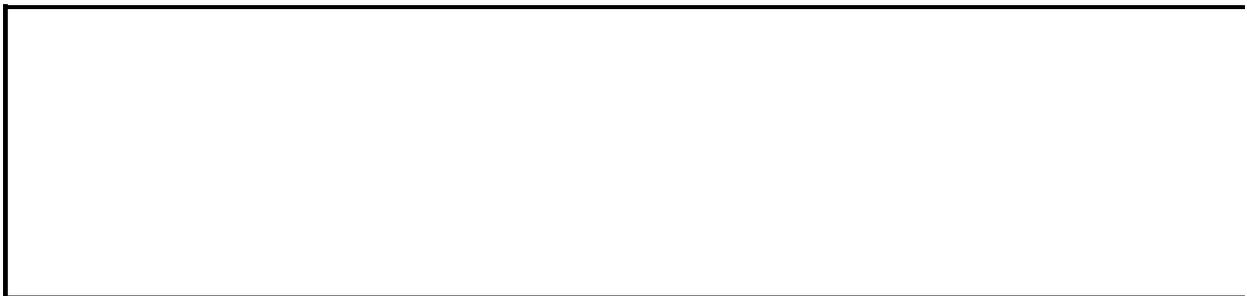
# PersonalViewPrintSettings Property

**True** if print settings are included in the user's personal view of the shared workbook. Read-write **Boolean**.

## Example

This example removes print settings and filter and sort settings from the user's personal view of workbook two.

```
With Workbooks(2)  
    .PersonalViewListSettings = False  
    .PersonalViewPrintSettings = False  
End With
```



[Show All](#)

# Perspective Property

 [Perspective property as it applies to the \*\*ThreeDFormat\*\* object.](#)

Determines whether the extrusion appears in perspective. Read/write **MsoTriState**.

MsoTriState can be one of these MsoTriState constants.

**msoCTrue** Does not apply to this property.

**msoFalse** The extrusion is a parallel, or orthographic, projection— that is, the walls don't narrow toward a vanishing point.

**msoTriStateMixed**

**msoTriStateToggle**

**msoTrue** The extrusion appears in perspective— that is, the walls of the extrusion narrow toward a vanishing point.

*expression*.**Perspective**

*expression* Required. An expression that returns a **ThreeDFormat** object.

 [Perspective property as it applies to the \*\*Chart\*\* object.](#)

Returns or sets the perspective for the 3-D chart view. Must be between 0 and 100. This property is ignored if the **RightAngleAxes** property is **True**.  
Read/write **Long**.

*expression*.**Perspective**

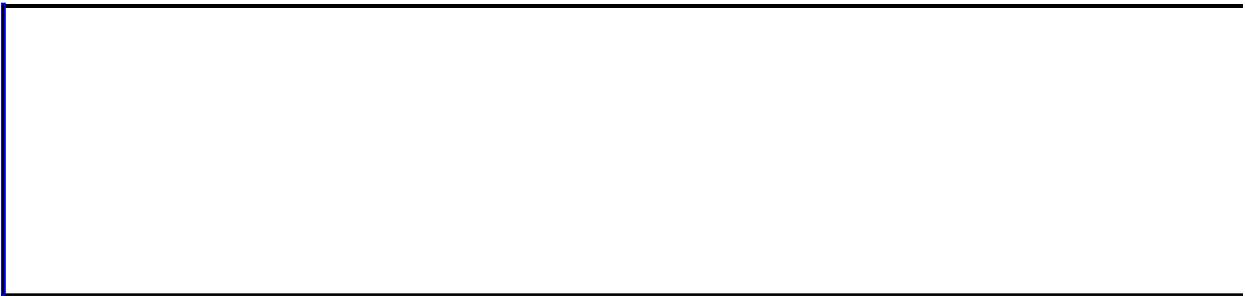
*expression* Required. An expression that returns a **Chart** object.

## Example

 [As it applies to the \*\*Chart\*\* object.](#)

This example sets the perspective of Chart1 to 70. The example should be run on a 3-D chart.

```
Charts("Chart1").RightAngleAxes = False  
Charts("Chart1").Perspective = 70
```



# Phonetic Property

Returns the [Phonetic](#) object, which contains information about a specific phonetic text string in a cell.

## Remarks

This property provides compatibility with earlier versions of Microsoft Excel. You should use **Phonetics**(*index*), where *index* is the index number of the phonetic text, to return a single **Phonetic** object.

For information about using phonetic worksheet functions in Microsoft Visual Basic, see [Using Microsoft Excel Worksheet Functions in Visual Basic](#).

## Example

This example sets the first phonetic text string in the active cell to "フリガナ".

```
ActiveCell.Phonetics(1).Text = "フリガナ"
```

To demonstrate compatibility with earlier versions of Microsoft Excel, this example hides the Furigana characters in cell C5.

```
Range("C5").Phonetic.Visible = False
```



# PhoneticCharacters Property

Returns or sets the phonetic text in the specified [Characters](#) object. Read/write **String**.

## Remarks

Instead of using this property, you should use the [Add](#) method of the [Phonetics](#) collection to add phonetic information to a cell, and use the [Text](#) property of the [Phonetic](#) object to return or set the phonetic text strings in a cell.

You can use this property only with **Characters** objects that are based on a single cell.

## Example

This example replaces the fourth character from the beginning of the text in the active cell with Furigana characters.

```
ActiveCell.Characters(1,3).PhoneticCharacters = "フリカサ"
```



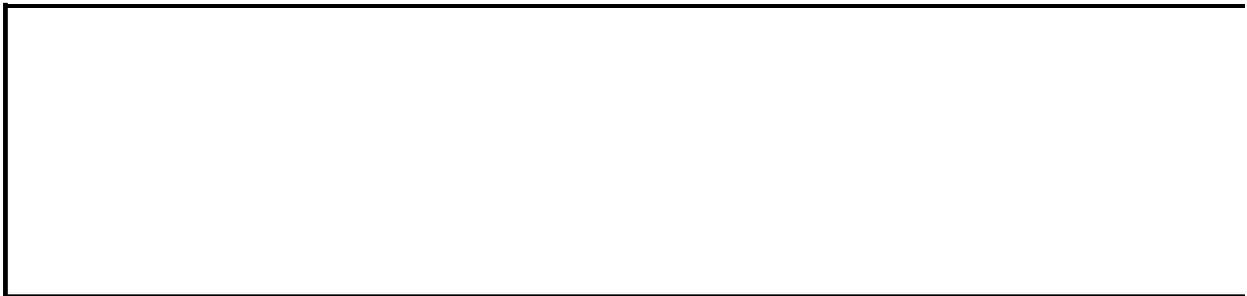
# Phonetics Property

Returns the [Phonetics](#) collection of the range. Read only **Phonetics**.

## Example

This example displays all of the **Phonetic** objects in the active cell.

```
Set objPhon = ActiveCell.Phonetics
With objPhon
  For Each objPhonItem in objPhon
    MsgBox "Phonetic object: " & .Text
  Next
End With
```



# PictureFormat Property

Returns a [PictureFormat](#) object that contains picture formatting properties for the specified shape. Applies to **Shape** or **ShapeRange** objects that represent pictures or OLE objects. Read-only.

## Example

This example sets the brightness and contrast for shape one on myDocument. Shape one must be a picture or an OLE object.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(1).PictureFormat
    .Brightness = 0.3
    .Contrast = .75
End With
```



[Show All](#)

# PictureType Property

 [PictureType property as it applies to the \*\*Point\*\* and \*\*Series\*\* objects.](#)

Returns or sets the way pictures are displayed on a column or bar picture chart. Read/write **XlChartPictureType**.

XlChartPictureType can be one of these XlChartPictureType constants.

**xlStack**

**xlStackScale**

**xlStretch**

*expression*.**PictureType**

*expression* Required. An expression that returns one of the above objects.

 [PictureType property as it applies to the \*\*LegendKey\*\* object.](#)

Returns or sets the way pictures are displayed on a legend key. Read/write **Long**.

*expression*.**PictureType**

*expression* Required. An expression that returns one of the above objects.

 [PictureType property as it applies to the \*\*Floor\*\* and \*\*Walls\*\* objects.](#)

Returns or sets the way pictures are displayed on the walls and faces of a 3-D chart. Read/write **Variant**.

*expression*.**PictureType**

*expression* Required. An expression that returns one of the above objects.

## Example

This example sets series one in Chart1 to stretch pictures. The example should be run on a 2-D column chart with picture data markers.

```
Charts("Chart1").SeriesCollection(1).PictureType = xlStretch
```



# PictureUnit Property

Returns or sets the unit for each picture on the chart if the [PictureType](#) property is set to **xlScale** (if not, this property is ignored). Read/write **Long**.

## Example

This example sets series one in Chart1 to stack pictures and uses each picture to represent five units. The example should be run on a 2-D column chart with picture data markers.

```
With Charts("Chart1").SeriesCollection(1)  
    .PictureType = xlScale  
    .PictureUnit = 5  
End With
```



# Pie3DGroup Property

Returns a [ChartGroup](#) object that represents the pie chart group on a 3-D chart.  
Read-only.

## Example

This example sets the 3-D pie group in Chart1 to use a different color for each data marker.

```
Charts("Chart1").Pie3DGroup.VaryByCategories = True
```



# PivotCell Property

Returns a **PivotCell** object that represents a cell in a PivotTable report.

*expression*.**PivotCell**

*expression* Required. An expression that returns a [Range](#) object.

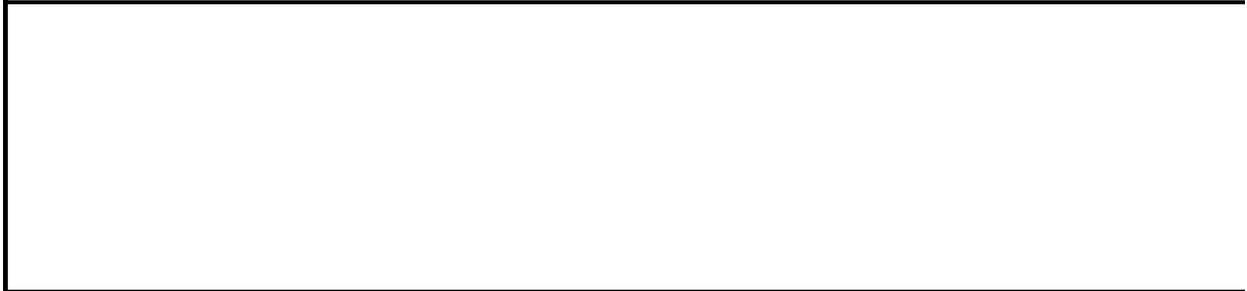
## Example

This example determines the name of the PivotTable the **PivotCell** object is located in and notifies the user. The example assumes that a PivotTable exists on the active worksheet and that cell A3 is located in the PivotTable.

```
Sub CheckPivotCell()
```

```
    'Determine the name of the PivotTable the PivotCell is located i  
    MsgBox "Cell A3 is located in PivotTable: " &  
        _ Application.Range("A3").PivotCell.Parent
```

```
End Sub
```



[Show All](#)

# PivotCellType Property

Returns one of the [XlPivotCellType](#) constants that identifies the PivotTable entity the cell corresponds to. Read-only.

XlPivotCellType can be one of these XlPivotCellType constants.

**xlPivotCellBlankCell** A structural blank cell in the PivotTable.

**xlPivotCellCustomSubtotal** A cell in the row or column area that is a custom subtotal.

**xlPivotCellDataField** A data field label (not the **Data** button).

**xlPivotCellDataPivotField** The **Data** button.

**xlPivotCellGrandTotal** A cell in a row or column area which is a grand total.

**xlPivotCellPageFieldItem** The cell that shows the selected item of a Page field.

**xlPivotCellPivotField** The button for a field (not the **Data** button).

**xlPivotCellPivotItem** A cell in the row or column area which is not a subtotal, grand total, custom subtotal, or blank line.

**xlPivotCellSubtotal** A cell in the row or column area which is a subtotal.

**xlPivotCellValue** Any cell in the data area (except a blank row).

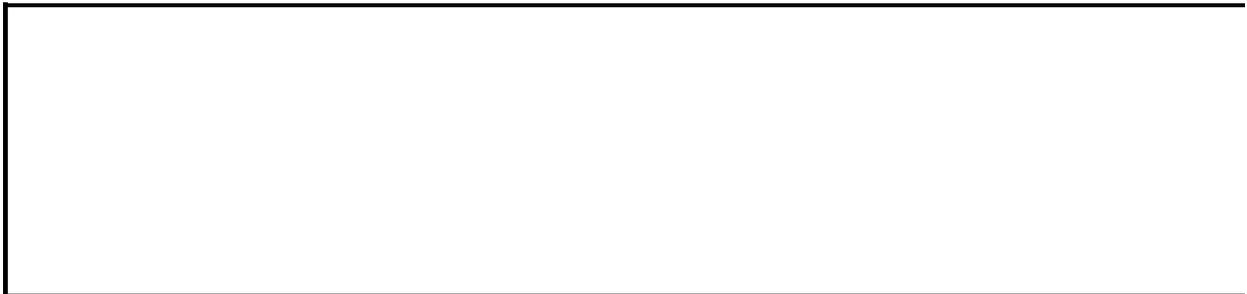
*expression*.**PivotCellType**

*expression* Required. An expression that returns a [PivotCell](#) object.

## Example

This example determines if cell A5 in the PivotTable is a data item and notifies the user. The example assumes a PivotTable exists on the active worksheet and cell A5 is contained in the PivotTable. If cell A5 is not in the PivotTable, the example handles the run-time error.

```
Sub CheckPivotCellType()  
  
    On Error GoTo Not_In_PivotTable  
  
    ' Determine if cell A5 is a data item in the PivotTable.  
    If Application.Range("A5").PivotCell.PivotCellType = xlPivotCell  
        MsgBox "The cell at A5 is a data item."  
    Else  
        MsgBox "The cell at A5 is not a data item."  
    End If  
    Exit Sub  
  
Not_In_PivotTable:  
    MsgBox "The chosen cell is not in a PivotTable."  
  
End Sub
```



# PivotField Property

Returns a [PivotField](#) object that represents the PivotTable field containing the upper-left corner of the specified range.

*expression*.**PivotField**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example displays the name of the PivotTable field that contains the active cell.

```
Worksheets("Sheet1").Activate  
MsgBox "The active cell is in the field " & _  
    ActiveCell.PivotField.Name
```



[Show All](#)

# PivotFields Property

Returns the [PivotFields](#) collection. This collection contains all PivotTable fields, including those that aren't currently visible on-screen. Read-only **PivotFields** object.

*expression*.**PivotFields**

*expression* Required. An expression that returns a **CubeField** object.

## Remarks

For [Online Analytical Processing \(OLAP\)](#) data sources, there are no hidden fields, and the object or collection that's returned reflects what's currently visible.

## Example

This example creates a list of all the PivotTable field names used in the first PivotChart report.

```
Set objNewSheet = Worksheets.Add  
objNewSheet.Activate  
intRow = 1  
For Each objPF In _  
    Charts("Chart1").PivotLayout.PivotFields  
    objNewSheet.Cells(intRow, 1).Value = objPF.Caption  
    intRow = intRow + 1  
Next objPF
```



[Show All](#)

# PivotFormulas Property

Returns a [PivotFormulas](#) object that represents the collection of formulas for the specified PivotTable report. Read-only.

*expression*.**PivotFormulas**

*expression* Required. An expression that returns a **PivotTable** object.

## Remarks

For [OLAP](#) data sources, this property returns an empty collection.

## Example

This example creates a list of formulas for PivotTable one.

```
For Each pf in ActiveSheet.PivotTables(1).PivotFormulas
    r = r + 1
    Cells(r, 1).Value = pf.Formula
Next
```



# PivotItem Property

Returns a [PivotItem](#) object that represents the PivotTable item containing the upper-left corner of the specified range.

*expression*.**PivotItem**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example displays the name of the PivotTable item that contains the active cell on Sheet1.

```
Worksheets("Sheet1").Activate  
MsgBox "The active cell is in the item " & _  
    ActiveCell.PivotItem.Name
```



# PivotLayout Property

Returns a [PivotLayout](#) object that represents the placement of fields in a PivotTable report and the placement of axes in a PivotChart report. Read-only.

## Remarks

If the chart you specify isn't a PivotChart report, the value of this property is **Nothing**.

## Example

This example creates a list of all the PivotTable field names used in the first PivotChart report.

```
Set objNewSheet = Worksheets.Add  
objNewSheet.Activate  
intRow = 1  
For Each objPF In _  
    Charts("Chart1").PivotLayout.PivotFields  
    objNewSheet.Cells(intRow, 1).Value = objPF.Caption  
    intRow = intRow + 1  
Next objPF
```



# PivotSelection Property

Returns or sets the PivotTable selection in standard PivotTable report selection format. Read/write **String**.

*expression*.**PivotSelection**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Setting this property is equivalent to calling the **PivotSelect** method with the *Mode* argument set to **xlDataAndLabel**.

## Example

This example selects the data and label for the salesperson named Bob in the first PivotTable report on worksheet one.

```
Worksheets(1).PivotTables(1).PivotSelection = "Salesman[Bob]"
```



[Show All](#)

# PivotSelectionStandard Property

Returns or sets a **String** indicating the PivotTable selection in standard [PivotTable report](#) format using English (United States) settings. Read/write.

*expression*.**PivotSelectionStandard**

*expression* Required. An expression that returns a [PivotTable](#) object.

## Remarks

The **PivotSelectionStandard** property is "international-friendly" whereas the **PivotSelection** method is not.

## Example

This example selects a field titled "1.57" in the PivotTable and inserts a blank column field before it. The example assumes a PivotTable exists on the active worksheet that contains a column field titled "1.57".

```
Sub CheckPivotSelectionStandard()  
    Dim pvtTable As PivotTable  
    Set pvtTable = ActiveSheet.PivotTables(1)  
    pvtTable.PivotSelectionStandard = "1.57"  
    Selection.Insert  
End Sub
```



# PivotTable Property

Returns a [PivotTable](#) object that represents the PivotTable report containing the upper-left corner of the specified range, or the PivotTable report associated with the PivotChart report.

*expression*.**PivotTable**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example sets the current page for the PivotTable report on Sheet1 to the page named "Canada."

```
Set pvtTable = Worksheets("Sheet1").Range("A3").PivotTable  
pvtTable.PivotFields("Country").CurrentPage = "Canada"
```

This example determines the PivotTable report associated with the Sales chart on the active worksheet, and then it sets the page named "Oregon" as the current page for the PivotTable report.

```
Set objPT = _  
    ActiveSheet.Charts("Sales").PivotLayout.PivotTable  
objPT.PivotFields("State").CurrentPageName = "Oregon"
```



# PivotTableSelection Property

**True** if PivotTable reports use structured selection. Read/write **Boolean**.

## Example

This example enables structured selection mode and then sets the first PivotTable report on worksheet one to allow only data to be selected.

```
Application.PivotTableSelection = True  
Worksheets(1).PivotTables(1).SelectionMode = xlDataOnly
```



# PixelsPerInch Property

Returns or sets the density (pixels per inch) of graphics images and table cells on a Web page. The range of settings is usually from 19 to 480, and common settings for popular screen sizes are 72, 96, and 120. The default setting is 96. Read/write **Long**.

## Remarks

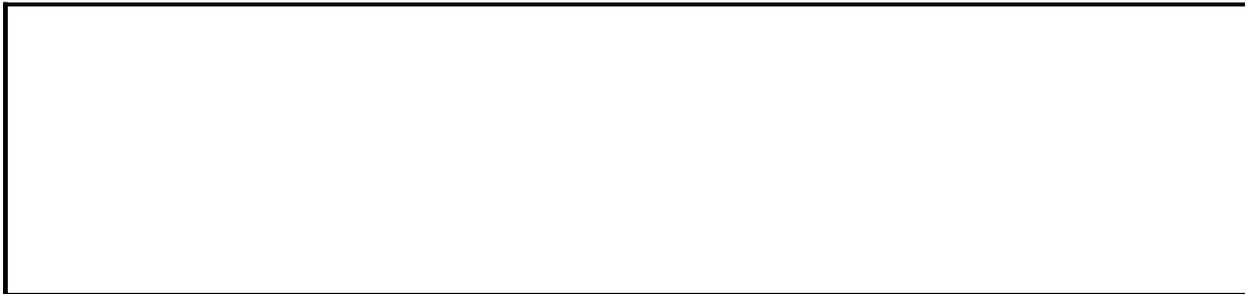
This property determines the size of the images and cells on the specified Web page relative to the size of text whenever you view the saved document in a Web browser. The physical dimensions of the resulting image or cell are the result of the original dimensions (in inches) multiplied by the number of pixels per inch.

You use the [ScreenSize](#) property to set the optimum screen size for the targeted Web browsers.

## Example

This example sets the pixel density depending on the target screen size of the browser. For 800x600 pixel screens, the density is 72 pixels per inch. For 1024x768 pixel screens, the density is 96 pixels per inch. For all other cases, use a density of 120 pixels per inch.

```
With Application.DefaultWebOptions
  Select Case .ScreenSize
    Case msoScreenSize800x600
      .PixelsPerInch = 72
    Case msoScreenSize1024x768
      .PixelsPerInch = 96
    Case Else
      .PixelsPerInch = 120
  End Select
End With
```



[Show All](#)

# Placement Property

 [Placement property as it applies to the \*\*Shape\*\* object.](#)

Returns or sets the way the object is attached to the cells below it. Read/write **XIPlacement**.

XIPlacement can be one of these XIPlacement constants.

**xlFreeFloating**

**xlMove**

**xlMoveAndSize**

*expression*.**Placement**

*expression* Required. An expression that returns one of the above objects.

 [Placement property as it applies to the \*\*ChartObject\*\*, \*\*ChartObjects\*\*, \*\*OLEObject\*\*, and \*\*OLEObjects\*\* objects.](#)

Returns or sets the way the object is attached to the cells below it. Read/write **Variant**.

*expression*.**Placement**

*expression* Required. An expression that returns one of the above objects.

## Example

This example sets embedded chart one on Sheet1 to be free-floating (it neither moves nor is sized with its underlying cells).

```
Worksheets("Sheet1").ChartObjects(1).Placement = xlFreeFloating
```



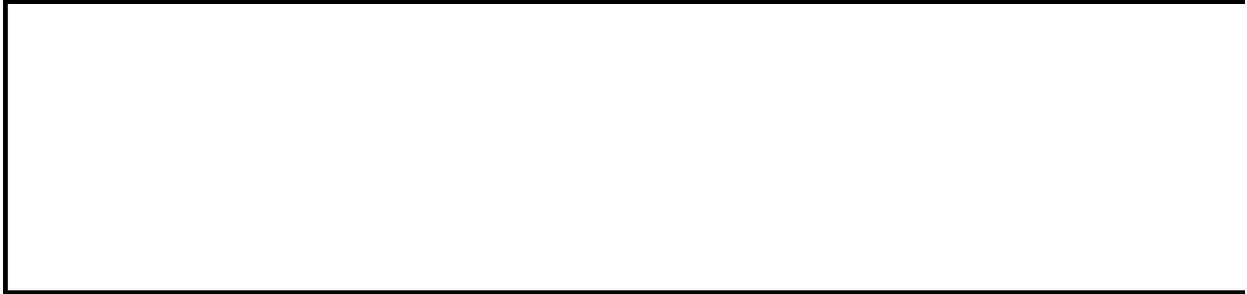
# PlotArea Property

Returns a [PlotArea](#) object that represents the plot area of a chart. Read-only.

## Example

This example sets the color of the plot area interior of Chart1 to cyan.

```
Charts("Chart1").PlotArea.Interior.ColorIndex = 8
```



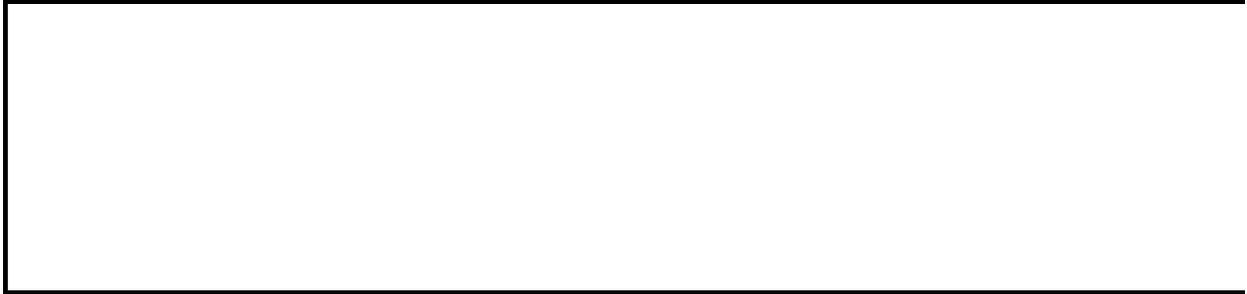
# PlotBy Property

Returns or sets the way columns or rows are used as data series on the chart. Can be one of the following **XlRowCol** constants: **xlColumns** or **xlRows**. Read/write **Long**. For PivotChart reports, this property is read-only and always returns **xlColumns**.

## Example

This example causes the embedded chart to plot data by columns.

```
Worksheets(1).ChartObjects(1).Chart.PlotBy = xlColumns
```



# PlotOrder Property

Returns or sets the plot order for the selected series within the chart group.  
Read/write **Long**.

## Remarks

You can set plot order only within a chart group (you cannot set the plot order for the entire chart if you have more than one chart type). A chart group is a collection of series with the same chart type.

Changing the plot order of one series will cause the plot orders of the other series in the chart group to be adjusted, as necessary.

## Example

This example makes series two in Chart1 appear third in the plot order. The example should be run on a 2-D column chart that contains three or more series.

```
Charts("Chart1").ChartGroups(1).SeriesCollection(2).PlotOrder = 3
```



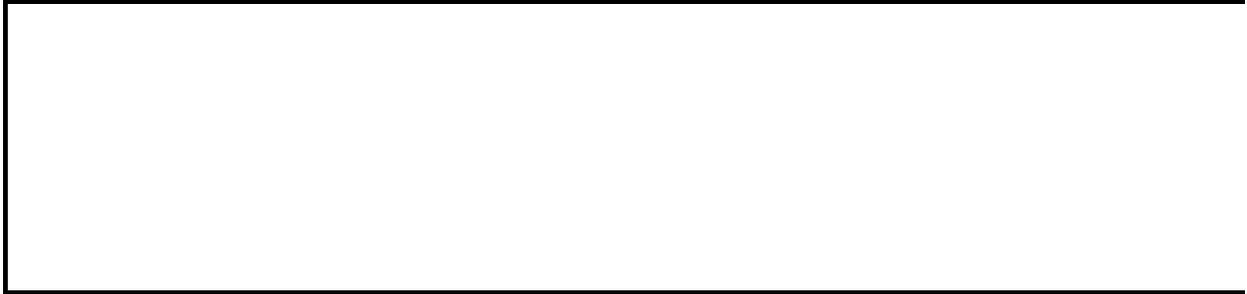
# PlotVisibleOnly Property

**True** if only visible cells are plotted. **False** if both visible and hidden cells are plotted. Read/write **Boolean**.

## Example

This example causes Microsoft Excel to plot only visible cells in Chart1.

```
Charts("Chart1").PlotVisibleOnly = True
```



[Show All](#)

# Points Property

Returns the position of the specified node as a [coordinate pair](#). Each coordinate is expressed in points. Read-only **Variant**.

## Remarks

This property is read-only. Use the [SetPosition](#) method to set the value of this property.

## Example

This example moves node two in shape three on myDocument to the right 200 points and down 300 points. Shape three must be a freeform drawing.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(3).Nodes
    pointsArray = .Item(2).Points
    currXvalue = pointsArray(1, 1)
    currYvalue = pointsArray(1, 2)
    .SetPosition 2, currXvalue + 200, currYvalue + 300
End With
```



[Show All](#)

# Position Property

 [Position property as it applies to the \*\*DataLabel\*\* and \*\*DataLabels\*\* objects.](#)

Returns or sets the position of the data label. Read/write [XLDataLabelPosition](#).

XLDataLabelPosition can be one of these XLDataLabelPosition constants.

**xLLabelPositionAbove**

**xLLabelPositionBestFit**

**xLLabelPositionCustom**

**xLLabelPositionInsideEnd**

**xLLabelPositionMixed**

**xLLabelPositionRight**

**xLLabelPositionBelow**

**xLLabelPositionCenter**

**xLLabelPositionInsideBase**

**xLLabelPositionLeft**

**xLLabelPositionOutsideEnd**

*expression*.**Position**

*expression* Required. An expression that returns one of the above objects.

 [Position property as it applies to the \*\*Legend\*\* object.](#)

Returns or sets the position of the legend on the chart. Read/write [XLLegendPosition](#).

XLLegendPosition can be one of these XLLegendPosition constants.

**xLegendPositionCorner**

**xLegendPositionRight**

**xLegendPositionTop**

**xLegendPositionBottom**

## **xlLegendPositionLeft**

### *expression*.**Position**

*expression* Required. An expression that returns one of the above objects.

[Position property as it applies to the \*\*CubeField\*\* and \*\*PivotItem\*\* objects.](#)

Position of the item in its field, if the item is currently showing. For a **CubeFields** collection, this is the position of the hierarchy field on the PivotTable report when it's dragged from the field well. Read/write **Long**.

### *expression*.**Position**

*expression* Required. An expression that returns one of the above objects.

[Position property as it applies to the \*\*PivotField\*\* object.](#)

Position of the field (first, second, third, and so on) among all the fields in its orientation (Rows, Columns, Pages, Data). Read/write **Variant**.

### *expression*.**Position**

*expression* Required. An expression that returns one of the above objects.

## Example

This example moves the chart legend to the bottom of the chart.

```
Charts(1).Legend.Position = xlLegendPositionBottom
```

This example displays the position number of the PivotTable item that contains the active cell.

```
Worksheets("Sheet1").Activate  
MsgBox "The active item is in position number " & _  
    ActiveCell.PivotItem.Position
```



# PostText Property

Some of the content in this topic may not be applicable to some languages.

Returns or sets the string used with the post method of inputting data into a Web server to return data from a Web query. Read/write **String**.

## Remarks

Microsoft Excel includes sample Web queries that you can modify by changing the HTML code by using WordPad or another text editor. You can find these samples in the Queries folder where you installed Microsoft Office.

---

---

# Precedents Property

Returns a [Range](#) object that represents all the precedents of a cell. This can be a multiple selection (a union of **Range** objects) if there's more than one precedent. Read-only.

## Example

This example selects the precedents of cell A1 on Sheet1.

```
Worksheets("Sheet1").Activate  
Range("A1").Precedents.Select
```



# PrecisionAsDisplayed Property

**True** if calculations in this workbook will be done using only the precision of the numbers as they're displayed. Read/write **Boolean**.

## Example

This example causes calculations in the active workbook to use only the precision of the numbers as they're displayed.

```
ActiveWorkbook.PrecisionAsDisplayed = True
```



[Show All](#)

# Prefix Property

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Returns a **String** that represents the prefix for the specified XML [namespace](#).  
Read-only.

*expression*.**Prefix**

*expression* Required. An expression that returns an [XmlNamespace](#) object.



# PrefixCharacter Property

Returns the prefix character for the cell. Read-only **Variant**.

## Remarks

If the [TransitionNavigKeys](#) property is **False**, this prefix character will be ' for a text label, or blank. If the **TransitionNavigKeys** property is **True**, this character will be ' for a left-justified label, " for a right-justified label, ^ for a centered label, \ for a repeated label, or blank.

## Example

This example displays the prefix character for cell A1 on Sheet1.

```
MsgBox "The prefix character is " & _  
Worksheets("Sheet1").Range("A1").PrefixCharacter
```

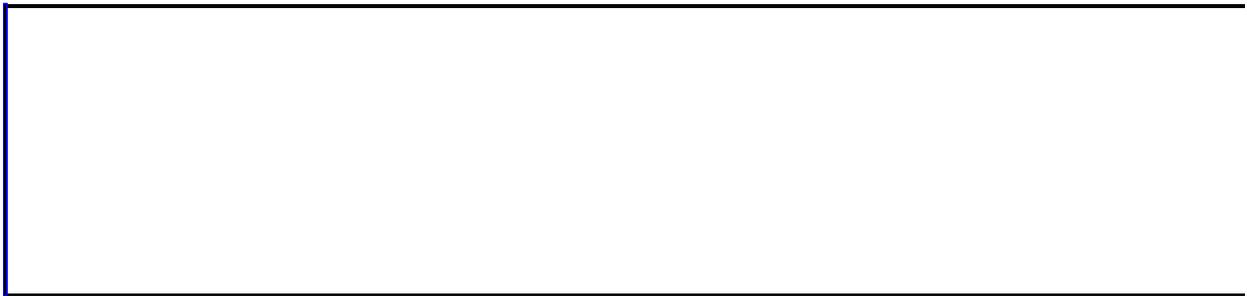


# PresentInPane Property

Returns a **Boolean** value that represents whether a smart document control is currently displayed in the **Document Actions** task pane. Read-only.

*expression*.**PresentInPane**

*expression* Required. An expression that returns a [SmartTagAction](#) object.



# PreserveColumnFilter Property

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Returns or sets whether filtering is preserved when the specified XML map is refreshed. Read/write **Boolean**.

*expression*.**PreserveColumnFilter**

*expression* Required. An expression that returns an [XmlMap](#) object.



# PreserveColumnInfo Property

**True** if column sorting, filtering, and layout information is preserved whenever a query table is refreshed. The default value is **False**. Read/write **Boolean**.

## Remarks

This property has an effect only when the query table is using a database connection.

You can set this property to **False** for compatibility with earlier versions of Microsoft Excel.

## Example

This example preserves column sorting, filtering, and layout information for compatibility with earlier versions of Microsoft Excel.

```
Dim cnnConnect As ADODB.Connection
Dim rstRecordset As ADODB.Recordset

Set cnnConnect = New ADODB.Connection
cnnConnect.Open "Provider=SQLOLEDB;" & _
    "Data Source=srvdata;" & _
    "User ID=wadet;Password=4me2no;"

Set rstRecordset = New ADODB.Recordset
rstRecordset.Open _
    Source:="Select Name, Quantity, Price From Products", _
    ActiveConnection:=cnnConnect, _
    CursorType:=adOpenDynamic, _
    LockType:=adLockReadOnly, _
    Options:=adCmdText

With ActiveSheet.QueryTables.Add( _
    Connection:=rstRecordset, _
    Destination:=Range("A1"))
    .Name = "Contact List"
    .FieldNames = True
    .RowNumbers = False
    .FillAdjacentFormulas = False
    .PreserveFormatting = True
    .RefreshOnFileOpen = False
    .BackgroundQuery = True
    .RefreshStyle = xlInsertDeleteCells
    .SavePassword = True
    .SaveData = True
    .AdjustColumnWidth = True
    .RefreshPeriod = 0
    .PreserveColumnInfo = True
    .Refresh BackgroundQuery:=False
End With
```



# PreserveFormatting Property

For PivotTable reports, this property is **True** if formatting is preserved when the report is refreshed or recalculated by operations such as pivoting, sorting, or changing page field items.

For query tables, this property is **True** if any formatting common to the first five rows of data are applied to new rows of data in the query table. Unused cells aren't formatted. The property is **False** if the last AutoFormat applied to the query table is applied to new rows of data. The default value is **True** (unless the query table was created in Microsoft Excel 97 and the [HasAutoFormat](#) property is **True**, in which case **PreserveFormatting** is **False**).

Read/write **Boolean**.

## Remarks

For database query tables, the default formatting setting is **xlSimple**.

The new AutoFormat style is applied to the query table when the table is refreshed. The AutoFormat is reset to **None** whenever **PreserveFormatting** is set to **False**. As a result, any AutoFormat that's set before **PreserveFormatting** is set to **False** and before the query table is refreshed doesn't take effect, and the resulting query table has no formatting applied to it.

## Example

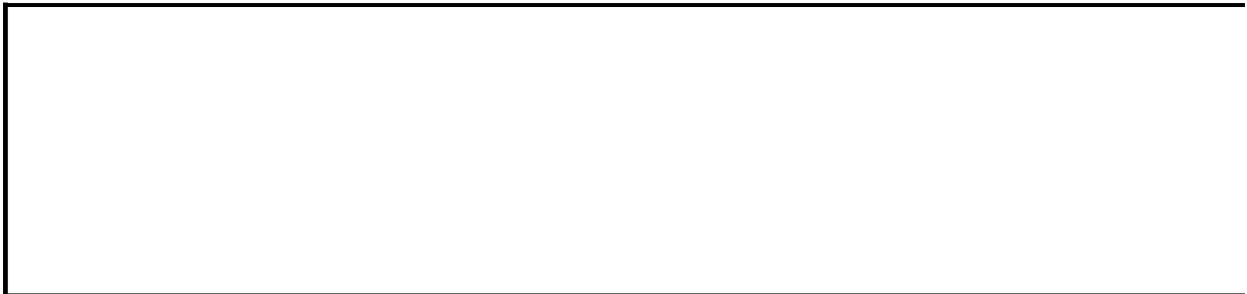
This example preserves the formatting of the first PivotTable report on worksheet one.

```
Worksheets(1).PivotTables("Pivot1").PreserveFormatting = True
```

This example demonstrates how setting **PreserveFormatting** to **False** causes the AutoFormat to be set to **xlRangeAutoFormatNone** instead of the specified **xlRangeAutoFormatColor1** format.

```
With Workbooks(1).Worksheets(1).QueryTables(1)  
    .Range.AutoFormat = xlRangeAutoFormatColor1  
    .PreserveFormatting = False  
    .Refresh
```

End With



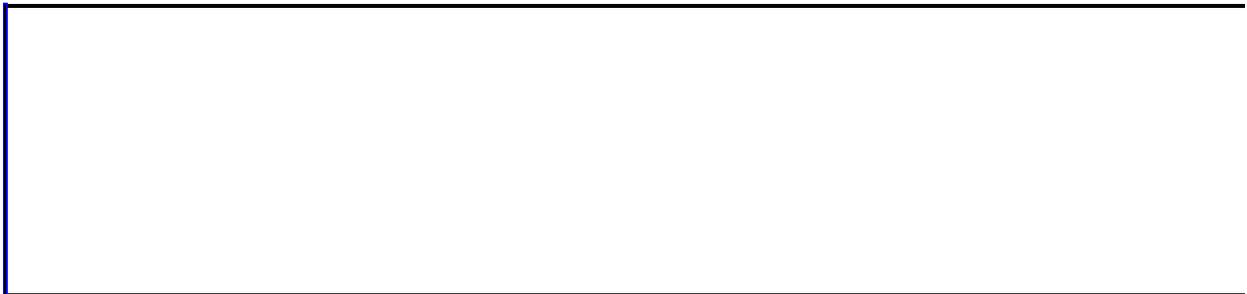
# PreserveNumberFormatting Property

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

**True** if number formatting on cells mapped to the specified XML schema map will be preserved when the schema map is refreshed. The default value is **False**.  
Read/write **Boolean**.

*expression*.**PreserveNumberFormatting**

*expression* Required. An expression that returns an [XmlMap](#) object.



[Show All](#)

# PresetExtrusionDirection Property

Returns the direction that the extrusion's sweep path takes away from the extruded shape (the front face of the extrusion). Read-only

[MsoPresetExtrusionDirection](#).

MsoPresetExtrusionDirection can be one of these MsoPresetExtrusionDirection constants.

**msoExtrusionTop**

**msoExtrusionTopRight**

**msoExtrusionBottom**

**msoExtrusionBottomLeft**

**msoExtrusionBottomRight**

**msoExtrusionLeft**

**msoExtrusionNone**

**msoExtrusionRight**

**msoExtrusionTopLeft**

**msoPresetExtrusionDirectionMixed**

*expression*.**PresetExtrusionDirection**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

This property is read-only. To set the value of this property, use the [SetExtrusionDirection](#) method.

## Example

This example changes each extrusion on myDocument that extends toward the upper-left corner of the extrusion's front face to an extrusion that extends toward the lower-right corner of the front face.

```
Set myDocument = Worksheets(1)
For Each s In myDocument.Shapes
  With s.ThreeD
    If .PresetExtrusionDirection = msoExtrusionTopLeft Then
      .SetExtrusionDirection msoExtrusionBottomRight
    End If
  End With
Next
```



[Show All](#)

# PresetGradientType Property

Returns the preset gradient type for the specified fill. Read-only [MsoPresetGradientType](#).

MsoPresetGradientType can be one of these MsoPresetGradientType constants.

**msoGradientBrass**

**msoGradientChrome**

**msoGradientDaybreak**

**msoGradientEarlySunset**

**msoGradientFog**

**msoGradientGoldII**

**msoGradientLateSunset**

**msoGradientMoss**

**msoGradientOcean**

**msoGradientPeacock**

**msoGradientRainbowII**

**msoGradientSilver**

**msoGradientWheat**

**msoPresetGradientMixed**

**msoGradientCalmWater**

**msoGradientChromeII**

**msoGradientDesert**

**msoGradientFire**

**msoGradientGold**

**msoGradientHorizon**

**msoGradientMahogany**

**msoGradientNightfall**

**msoGradientParchment**

**msoGradientRainbow**

**msoGradientSapphire**

## *expression*.**PresetGradientType**

*expression* Required. An expression that returns one of the objects in the Applies To list.

Use the **PresetGradient** method to set the preset gradient type for the fill.

## Example

This example sets the fill format for chart two to the same style used for chart one.

```
Set c1f = Charts(1).ChartArea.Fill
If c1f.Type = msoFillGradient Then
    With Charts(2).ChartArea.Fill
        .Visible = True
        .PresetGradient c1f.GradientStyle, _
            c1f.GradientVariant, c1f.PresetGradientType
    End With
End If
```



[Show All](#)

# PresetLightingDirection Property

Returns or sets the position of the light source relative to the extrusion.

Read/write [MsoPresetLightingDirection](#).

MsoPresetLightingDirection can be one of these MsoPresetLightingDirection constants.

**msoLightingBottom**

**msoLightingBottomLeft**

**msoLightingBottomRight**

**msoLightingLeft**

**msoLightingNone**

**msoLightingRight**

**msoLightingTop**

**msoLightingTopLeft**

**msoLightingTopRight**

**msoPresetLightingDirectionMixed**

*expression*.**PresetLightingDirection**

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Note** You won't see the lighting effects you set if the extrusion has a wire frame surface.

## Example

This example specifies that the extrusion for shape one on myDocument extend toward the top of the shape and that the lighting for the extrusion come from the left.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(1).ThreeD
    .Visible = True
    .SetExtrusionDirection msoExtrusionTop
    .PresetLightingDirection = msoLightingLeft
End With
```



[Show All](#)

# PresetLightingSoftness Property

Returns or sets the intensity of the extrusion lighting. Read/write [MsoPresetLightingSoftness](#).

MsoPresetLightingSoftness can be one of these MsoPresetLightingSoftness constants.

**msoLightingBright**

**msoLightingDim**

**msoLightingNormal**

**msoPresetLightingSoftnessMixed**

*expression*.**PresetLightingSoftness**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example specifies that the extrusion for shape one on myDocument be lit brightly from the left.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(1).ThreeD
    .Visible = True
    .PresetLightingSoftness = msoLightingBright
    .PresetLightingDirection = msoLightingLeft
End With
```



[Show All](#)

# PresetMaterial Property

Returns or sets the extrusion surface material. Read/write [MsoPresetMaterial](#).

MsoPresetMaterial can be one of these MsoPresetMaterial constants.

**msoMaterialMatte**

**msoMaterialMetal**

**msoMaterialPlastic**

**msoMaterialWireFrame**

**msoPresetMaterialMixed**

*expression*.**PresetMaterial**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example specifies that the extrusion surface for shape one in myDocument be wire frame.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(1).ThreeD
    .Visible = True
    .PresetMaterial = msoMaterialWireFrame
End With
```



[Show All](#)

# PresetShape Property

Returns or sets the shape of the specified WordArt. Read/write [MsoPresetTextEffectShape](#).

MsoPresetTextEffectShape can be one of these MsoPresetTextEffectShape constants.

**msoTextEffectShapeArchDownCurve**

**msoTextEffectShapeArchDownPour**

**msoTextEffectShapeArchUpCurve**

**msoTextEffectShapeArchUpPour**

**msoTextEffectShapeButtonCurve**

**msoTextEffectShapeButtonPour**

**msoTextEffectShapeCanDown**

**msoTextEffectShapeCanUp**

**msoTextEffectShapeCascadeDown**

**msoTextEffectShapeCascadeUp**

**msoTextEffectShapeChevronDown**

**msoTextEffectShapeChevronUp**

**msoTextEffectShapeCircleCurve**

**msoTextEffectShapeCirclePour**

**msoTextEffectShapeCurveDown**

**msoTextEffectShapeCurveUp**

**msoTextEffectShapeDeflate**

**msoTextEffectShapeDeflateBottom**

**msoTextEffectShapeDeflateInflateDeflate**

**msoTextEffectShapeDoubleWave1**

**msoTextEffectShapeFadeDown**

**msoTextEffectShapeFadeRight**

**msoTextEffectShapeInflate**

**msoTextEffectShapeInflateTop**

**msoTextEffectShapePlainText**

**msoTextEffectShapeRingOutside**  
**msoTextEffectShapeSlantUp**  
**msoTextEffectShapeTriangleDown**  
**msoTextEffectShapeWave1**  
**msoTextEffectShapeDeflateInflate**  
**msoTextEffectShapeDeflateTop**  
**msoTextEffectShapeDoubleWave2**  
**msoTextEffectShapeFadeLeft**  
**msoTextEffectShapeFadeUp**  
**msoTextEffectShapeInflateBottom**  
**msoTextEffectShapeMixed**  
**msoTextEffectShapeRingInside**  
**msoTextEffectShapeSlantDown**  
**msoTextEffectShapeStop**  
**msoTextEffectShapeTriangleUp**  
**msoTextEffectShapeWave2**

*expression*.**PresetShape**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Setting the [PresetTextEffect](#) property automatically sets the **PresetShape** property.

## Example

This example sets the shape of all WordArt on myDocument to a chevron whose center points down.

```
Set myDocument = Worksheets(1)
For Each s In myDocument.Shapes
    If s.Type = msoTextEffect Then
        s.TextEffect.PresetShape = msoTextEffectShapeChevronDown
    End If
Next
```



[Show All](#)

# PresetTextEffect Property

Returns or sets the style of the specified WordArt. The values for this property correspond to the formats in the **WordArt Gallery** dialog box (numbered from left to right, top to bottom). Read/write [MsoPresetTextEffect](#).

MsoPresetTextEffect can be one of these MsoPresetTextEffect constants.

**msoTextEffect1**

**msoTextEffect10**

**msoTextEffect11**

**msoTextEffect12**

**msoTextEffect13**

**msoTextEffect14**

**msoTextEffect15**

**msoTextEffect16**

**msoTextEffect17**

**msoTextEffect18**

**msoTextEffect19**

**msoTextEffect2**

**msoTextEffect20**

**msoTextEffect21**

**msoTextEffect22**

**msoTextEffect23**

**msoTextEffect24**

**msoTextEffect25**

**msoTextEffect26**

**msoTextEffect27**

**msoTextEffect28**

**msoTextEffect29**

**msoTextEffect3**

**msoTextEffect30**

**msoTextEffect4**

**msoTextEffect5**

**msoTextEffect6**

**msoTextEffect7**

**msoTextEffect8**

**msoTextEffect9**

**msoTextEffectMixed**

*expression*.**PresetTextEffect**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Setting the **PresetTextEffect** property automatically sets many other formatting properties of the specified shape.

## Example

This example sets the style for all WordArt on myDocument to the first style listed in the **WordArt Gallery** dialog box.

```
Set myDocument = Worksheets(1)
For Each s In myDocument.Shapes
    If s.Type = msoTextEffect Then
        s.TextEffect.PresetTextEffect = msoTextEffect1
    End If
Next
```



[Show All](#)

# PresetTexture Property

Returns the preset texture for the specified fill. Read-only [MsoPresetTexture](#).

MsoPresetTexture can be one of these MsoPresetTexture constants.

**msoPresetTextureMixed**

**msoTextureBouquet**

**msoTextureCanvas**

**msoTextureDenim**

**msoTextureGranite**

**msoTextureMediumWood**

**msoTextureOak**

**msoTexturePapyrus**

**msoTexturePinkTissuePaper**

**msoTextureRecycledPaper**

**msoTextureStationery**

**msoTextureWaterDroplets**

**msoTextureWovenMat**

**msoTextureBlueTissuePaper**

**msoTextureBrownMarble**

**msoTextureCork**

**msoTextureFishFossil**

**msoTextureGreenMarble**

**msoTextureNewsprint**

**msoTexturePaperBag**

**msoTextureParchment**

**msoTexturePurpleMesh**

**msoTextureSand**

**msoTextureWalnut**

**msoTextureWhiteMarble**

*expression*.PresetTexture

*expression* Required. An expression that returns one of the objects in the Applies To list.

Use the **PresetTextured** method to set the preset texture for the fill.

## Example

This example sets the fill format for chart two to the same style used for chart one.

```
Set c1f = Charts(1).ChartArea.Fill
If c1f.Type = msoFillTextured Then
    With Charts(2).ChartArea.Fill
        .Visible = True
        If c1f.TextureType = msoTexturePreset Then
            .PresetTextured c1f.PresetTexture
        Else
            .UserTextured c1f.TextureName
        End If
    End With
End If
```



[Show All](#)

# PresetThreeDFormat Property

Returns the preset extrusion format. Each preset extrusion format contains a set of preset values for the various properties of the extrusion. If the extrusion has a custom format rather than a preset format, this property returns **msoPresetThreeDFormatMixed**. The values for this property correspond to the options (numbered from left to right, top to bottom) displayed when you click the **3-D** button on the **Drawing** toolbar. Read-only [MsoPresetThreeDFormat](#).

MsoPresetThreeDFormat can be one of these MsoPresetThreeDFormat constants.

**msoThreeD1**

**msoThreeD11**

**msoThreeD13**

**msoThreeD15**

**msoThreeD17**

**msoThreeD19**

**msoThreeD20**

**msoThreeD4**

**msoThreeD6**

**msoThreeD8**

**msoPresetThreeDFormatMixed**

**msoThreeD10**

**msoThreeD12**

**msoThreeD14**

**msoThreeD16**

**msoThreeD18**

**msoThreeD2**

**msoThreeD3**

**msoThreeD5**

**msoThreeD7**

**msoThreeD9**

*expression*.**PresetThreeDFormat**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

This property is read-only. To set the preset extrusion format, use the [SetThreeDFormat](#) method.

## Example

This example sets the extrusion format for shape one on myDocument to 3D Style 12 if the shape initially has a custom extrusion format.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(1).ThreeD
    If .PresetThreeDFormat = msoPresetThreeDFormatMixed Then
        .SetThreeDFormat msoThreeD12
    End If
End With
```



# Previous Property

Returns a [Chart](#), [Range](#), or [Worksheet](#) object that represents the previous sheet or cell. Read-only.

## Remarks

If the object is a range, this property emulates pressing SHIFT+TAB; unlike the key combination, however, the property returns the previous cell without selecting it.

On a protected sheet, this property returns the previous unlocked cell. On an unprotected sheet, this property always returns the cell immediately to the left of the specified cell.

## Example

This example selects the previous unlocked cell on Sheet1. If Sheet1 is unprotected, this is the cell immediately to the left of the active cell.

```
Worksheets("Sheet1").Activate  
ActiveCell.Previous.Select
```



# PreviousSelections Property

Returns an array of the last four ranges or names selected. Each element in the array is a [Range](#) object. Read-only **Variant**.

*expression*.**PreviousSelections**(*Index*)

*expression* Optional. An expression that returns an **Application** object.

*Index* Optional **Variant**. The index number (from 1 to 4) of the previous range or name.

## Remarks

Each time you go to a range or cell by using the **Name** box or the **Go To** command (**Edit** menu), or each time a macro calls the [Goto](#) method, the previous range is added to this array as element number 1, and the other items in the array are moved down.

## Example

This example displays the cell addresses of all items in the array of previous selections. If there are no previous selections, the **LBound** function returns an error. This error is trapped, and a message box appears.

```
On Error GoTo noSelections
For i = LBound(Application.PreviousSelections) To _
    UBound(Application.PreviousSelections)
    MsgBox Application.PreviousSelections(i).Address
Next i
Exit Sub
On Error GoTo 0

noSelections:
    MsgBox "There are no previous selections"
```



# PrintArea Property

Returns or sets the range to be printed, as a string using A1-style references in the language of the macro. Read/write **String**.

## Remarks

Set this property to **False** or to the empty string ("") to set the print area to the entire sheet.

This property applies only to worksheet pages.

## Example

This example sets the print area to cells A1:C5 on Sheet1.

```
Worksheets("Sheet1").PageSetup.PrintArea = "$A$1:$C$5"
```

This example sets the print area to the current region on Sheet1. Note that you use the **Address** property to return an A1-style address.

```
Worksheets("Sheet1").Activate  
ActiveSheet.PageSetup.PrintArea = _  
    ActiveCell.CurrentRegion.Address
```



[Show All](#)

# PrintComments Property

Returns or sets the way comments are printed with the sheet. Read/write [XLPrintLocation](#).

XLPrintLocation can be one of these XLPrintLocation constants.

**xlPrintInPlace**

**xlPrintNoComments**

**xlPrintSheetEnd**

*expression*.**PrintComments**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example causes comments to be printed as end notes when worksheet one is printed.

```
Worksheets(1).PageSetup.PrintComments = xlPrintSheetEnd
```



[Show All](#)

# PrintErrors Property

Sets or returns an [XlPrintErrors](#) constant specifying the type of print error displayed. This feature allows users to suppress the display of error values when printing a worksheet. Read/write .

XlPrintErrors can be one of these XlPrintErrors constants.

**xlPrintErrorsBlank**

**xlPrintErrorsDash**

**xlPrintErrorsDisplayed**

**xlPrintErrorsNA**

*expression*.**PrintErrors**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel uses a formula that returns an error in the active worksheet. The **PrintErrors** property is set to display dashes. A Print Preview window displays the dashes for the print error. This example assumes a printer driver has been installed.

```
Sub UsePrintErrors()  
  
    Dim wksOne As Worksheet  
  
    Set wksOne = Application.ActiveSheet  
  
    ' Create a formula that returns an error value.  
    Range("A1").Value = 1  
    Range("A2").Value = 0  
    Range("A3").Formula = "=A1/A2"  
  
    ' Change print errors to display dashes.  
    wksOne.PageSetup.PrintErrors = xlPrintErrorsDash  
  
    ' Use the Print Preview window to see the dashes used for print  
    ActiveWindow.SelectedSheets.PrintPreview  
  
End Sub
```



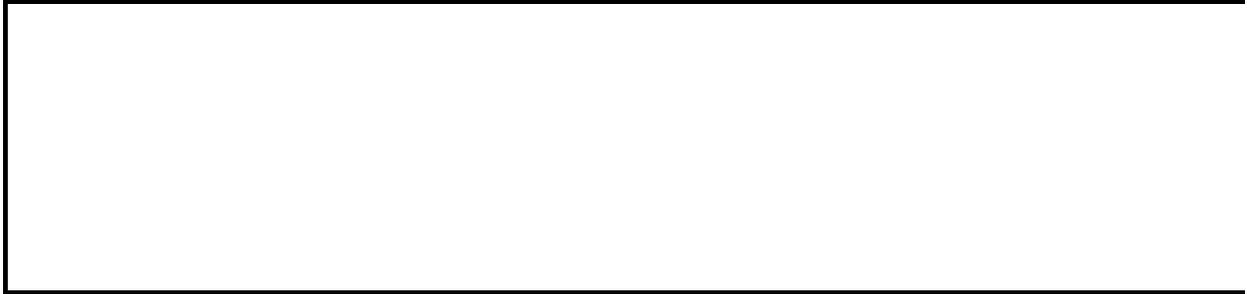
# PrintGridlines Property

**True** if cell gridlines are printed on the page. Applies only to worksheets.  
Read/write **Boolean**.

## Example

This example prints cell gridlines when Sheet1 is printed.

```
Worksheets("Sheet1").PageSetup.PrintGridlines = True
```



# PrintHeadings Property

**True** if row and column headings are printed with this page. Applies only to worksheets. Read/write **Boolean**.

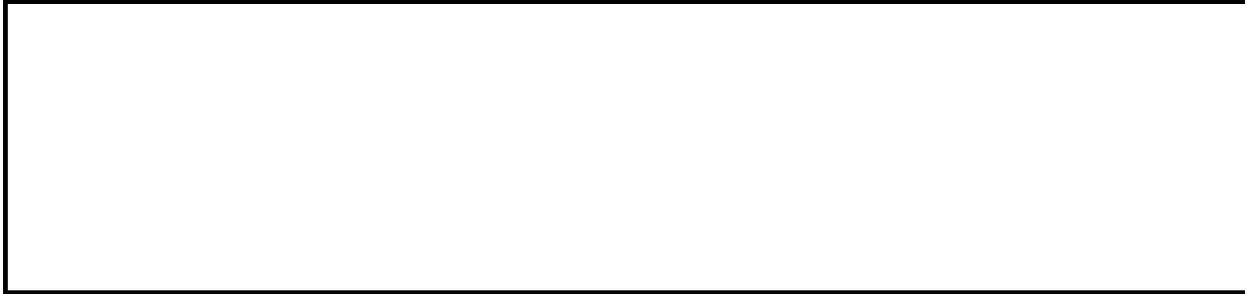
## Remarks

The [DisplayHeadings](#) property controls the on-screen display of headings.

## Example

This example turns off the printing of headings for Sheet1.

```
Worksheets("Sheet1").PageSetup.PrintHeadings = False
```



# PrintNotes Property

**True** if cell notes are printed as end notes with the sheet. Applies only to worksheets. Read/write **Boolean**.

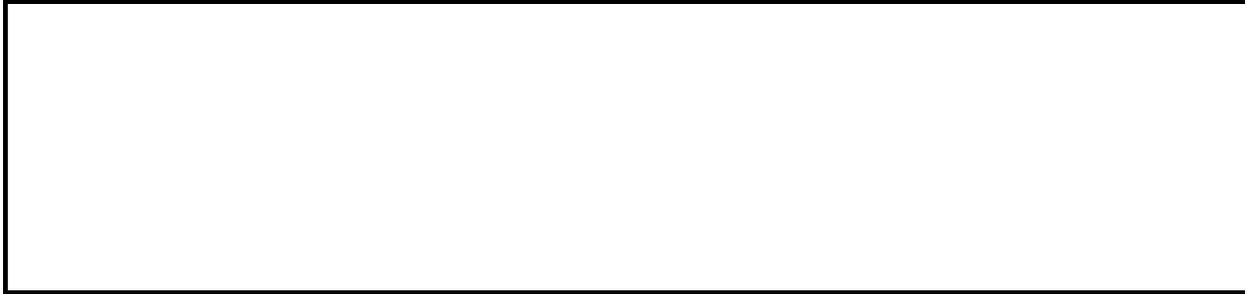
## Remarks

Use the **PrintComments** property to print comments as text boxes or end notes.

## Example

This example turns off the printing of notes.

```
Worksheets("Sheet1").PageSetup.PrintNotes = False
```



# PrintObject Property

**True** if the object will be printed when the document is printed. Read/write **Boolean**.

## Example

This example sets embedded chart one on Sheet1 to be printed with the worksheet.

```
Worksheets("Sheet1").ChartObjects(1).PrintObject = True
```



# PrintQuality Property

Returns or sets the print quality. Read/write **Variant**.

*expression*.**PrintQuality**(*Index*)

*expression* Required. An expression that returns a **PageSetup** object.

**Index** Optional **Variant**. Horizontal print quality (1) or vertical print quality (2). Some printers may not support vertical print quality. If you don't specify this argument, the **PrintQuality** property returns (or can be set to) a two-element array that contains both horizontal and vertical print quality.

## Example

This example sets the print quality on a printer with non-square pixels. The array specifies both horizontal and vertical print quality. This example may cause an error, depending on the printer driver you're using.

```
Worksheets("Sheet1").PageSetup.PrintQuality = Array(240, 140)
```

This example displays the current setting for horizontal print quality.

```
MsgBox "Horizontal Print Quality is " & _  
    Worksheets("Sheet1").PageSetup.PrintQuality(1)
```



# PrintSettings Property

**True** if print settings are included in the custom view. Read-only **Boolean**.

## Example

This example creates a list of the custom views in the active workbook and their print settings and row and column settings.

```
With Worksheets(1)
    .Cells(1,1).Value = "Name"
    .Cells(1,2).Value = "Print Settings"
    .Cells(1,3).Value = "RowColSettings"
    rw = 0
    For Each v In ActiveWorkbook.CustomViews
        rw = rw + 1
        .Cells(rw, 1).Value = v.Name
        .Cells(rw, 2).Value = v.PrintSettings
        .Cells(rw, 3).Value = v.RowColSettings
    Next
End With
```



# PrintTitleColumns Property

Returns or sets the columns that contain the cells to be repeated on the left side of each page, as a string in A1-style notation in the language of the macro.  
Read/write **String**.

## Remarks

If you specify only part of a column or columns, Microsoft Excel expands the range to full columns.

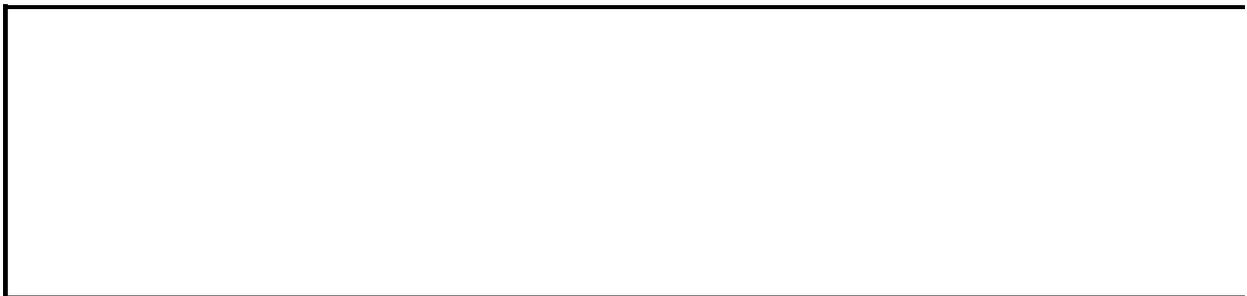
Set this property to **False** or to the empty string ("") to turn off title columns.

This property applies only to worksheet pages.

## Example

This example defines row three as the title row, and it defines columns one through three as the title columns.

```
Worksheets("Sheet1").Activate  
ActiveSheet.PageSetup.PrintTitleRows = ActiveSheet.Rows(3).Address  
ActiveSheet.PageSetup.PrintTitleColumns = _  
    ActiveSheet.Columns("A:C").Address
```



# PrintTitleRows Property

Returns or sets the rows that contain the cells to be repeated at the top of each page, as a string in A1-style notation in the language of the macro. Read/write **String**.

## Remarks

If you specify only part of a row or rows, Microsoft Excel expands the range to full rows.

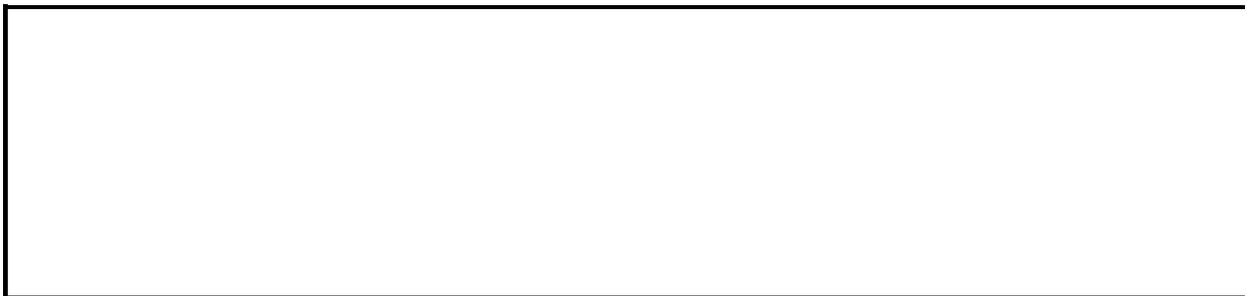
Set this property to **False** or to the empty string ("") to turn off title rows.

This property applies only to worksheet pages.

## Example

This example defines row three as the title row, and it defines columns one through three as the title columns.

```
Worksheets("Sheet1").Activate  
ActiveSheet.PageSetup.PrintTitleRows = ActiveSheet.Rows(3).Address  
ActiveSheet.PageSetup.PrintTitleColumns = _  
    ActiveSheet.Columns("A:C").Address
```



# PrintTitles Property

**True** if the print titles for the worksheet are set based on the PivotTable report. The row print titles are set to the rows that contain the PivotTable report's column field items. The column print titles are set to the columns that contain the row items. **False** if the print titles for the worksheet are used. The default value is **False**. Read/write **Boolean**.

## Remarks

The PivotTable report must be the only one in the print area. To set an indented format for a PivotTable report, use the [Format](#) method.

## Example

This example specifies that the print title set for the worksheet is printed when the fourth PivotTable report on the active worksheet is printed.

```
ActiveSheet.PivotTables("PivotTable4").PrintTitles = True
```



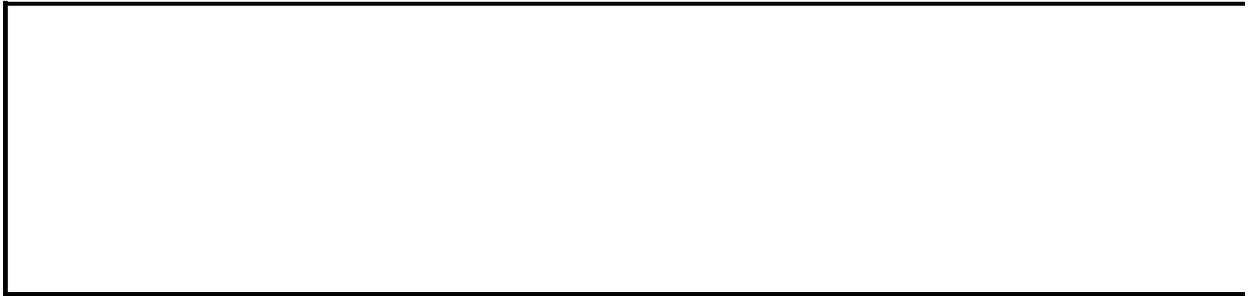
# ProductCode Property

Returns the globally unique identifier (GUID) for Microsoft Excel. Read-only **String**.

## Example

This example displays the globally unique identifier (GUID) for Microsoft Excel.

MsgBox Application.**ProductCode**



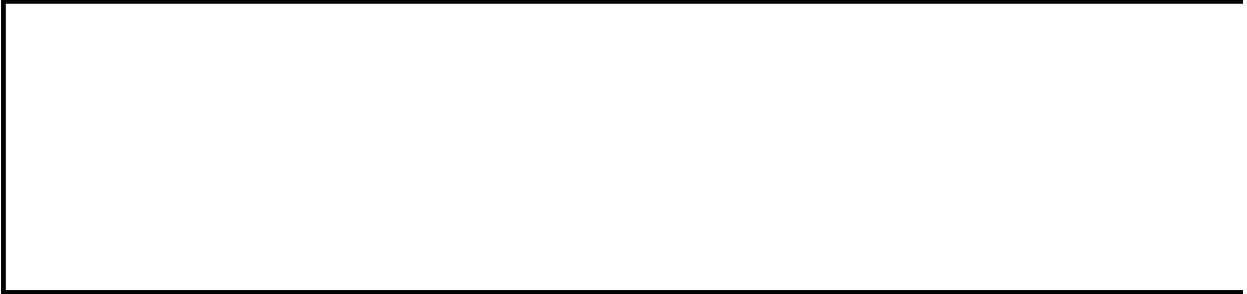
# PromptForSummaryInfo Property

**True** if Microsoft Excel asks for summary information when files are first saved.  
Read/write **Boolean**.

## Example

This example displays a prompt that asks for summary information when files are first saved.

```
Application.PromptForSummaryInfo = True
```



# PromptString Property

Returns the phrase that prompts the user for a parameter value in a parameter query. Read-only **String**.

## Example

This example modifies the parameter prompt string for query table one.

```
With Worksheets(1).QueryTables(1).Parameters(1)  
    .SetParam xlPrompt, "Please " & .PromptString  
End With
```



# Properties Property

Returns a **CustomProperties** object representing the properties for a smart tag.

*expression*.**Properties**

*expression* Required. An expression that returns one of the objects in the Applies To list.

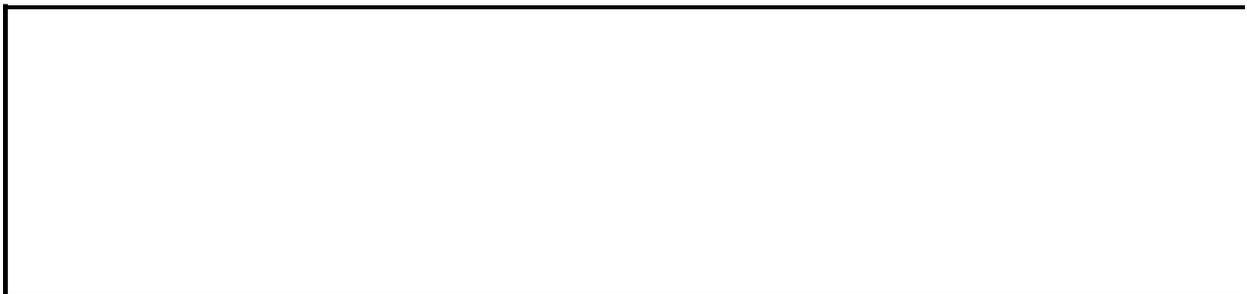
## Remarks

Use the **Add** Method with the **Properties** property to store extra metadata for a smart tag.

## Example

This example adds a smart tag to cell A1, then adds extra metadata called "Market" with the value of "Nasdaq" to the smart tag and then returns the value of the property to the user. This example assumes the host system is connected to the Internet.

```
Sub UseProperties()  
  
    Dim strLink As String  
    Dim strType As String  
  
    ' Define smart tag variables.  
    strLink = "urn:schemas-microsoft-com:smartrags#StockTickerSymbol  
    strType = "stockview"  
  
    ' Enable smart tags to be embedded and recognized.  
    ActiveWorkbook.SmartTagOptions.EmbedSmartTags = True  
    Application.SmartTagRecognizers.Recognize = True  
  
    ' Add a property for MSFT smart tag and define it's value.  
    Range("A1").SmartTags.Add(strLink).Properties.Add _  
        Name:="Market", Value:="Nasdaq"  
  
    ' Notify the user of the smart tag's value.  
    MsgBox Range("A1").SmartTags.Add(strLink).Properties("Market").V  
  
End Sub
```



[Show All](#)

# PropertyOrder Property

Valid only for PivotTable fields that are member property fields. Returns a **Long** indicating the display position of the member property within the cube field to which it belongs. Setting this property will rearrange the order of the properties for this cube field. This property is one-based. The allowable range is from one to the maximum number of member property fields being displayed for the hierarchy. Read/write.

*expression*.**PropertyOrder**

*expression* Required. An expression that returns a [PivotField](#) object.

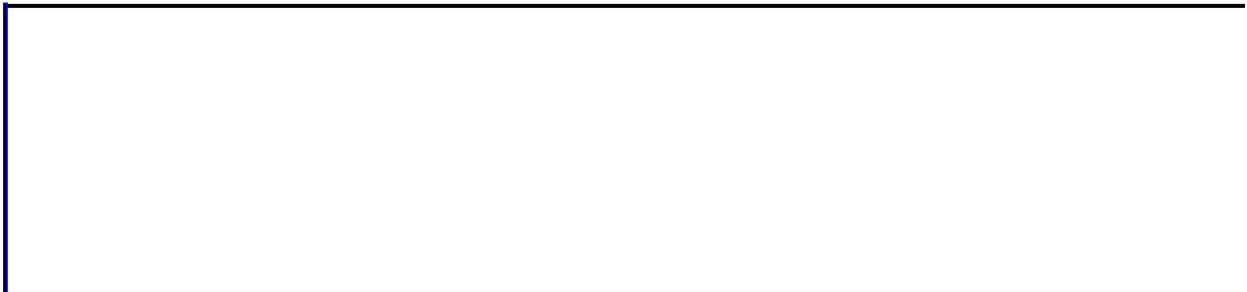
## Remarks

If the [IsMemberProperty](#) property is **False**, using the **PropertyOrder** property will create a run-time error.

## Example

This example determines if there are member properties in the fourth field and, if there are, displays the position of the member properties. Depending on the findings, Excel notifies the user. This example assumes that a PivotTable exists on the active worksheet and that it is based on an [Online Analytical Processing \(OLAP\)](#) data source.

```
Sub CheckPropertyOrder()  
  
    Dim pvtTable As PivotTable  
    Dim pvtField As PivotField  
  
    Set pvtTable = ActiveSheet.PivotTables(1)  
    Set pvtField = pvtTable.PivotFields(4)  
  
    ' Check for member properties and notify user.  
    If pvtField.IsMemberProperty = False Then  
        MsgBox "No member properties present."  
    Else  
        MsgBox "The property order of the members is: " & _  
            pvtField.PropertyOrder  
    End If  
  
End Sub
```



[Show All](#)

# PropertyParentField Property

Returns a **PivotField** object representing the field to which the properties in this field pertain.

*expression*.**PropertyParentField**

*expression* Required. An expression that returns a [PivotField](#) object.

## Remarks

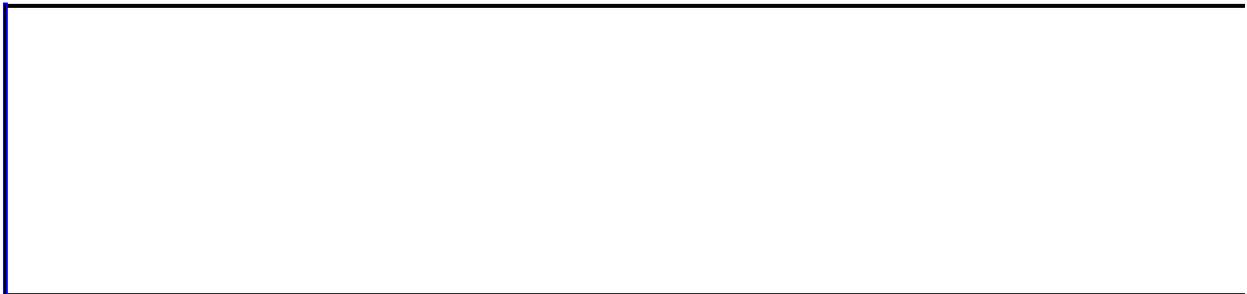
Valid only for fields that are member property fields.

If the [IsMemberProperty](#) property is **False**, using the **PropertyParentField** property will return a run-time error.

## Example

This example determines if there are member properties in the fourth field and, if there are, which fields the properties pertain to. Depending on the findings, Excel notifies the user. This example assumes that a PivotTable exists on the active worksheet and that it is based on an [Online Analytical Processing \(OLAP\)](#) data source.

```
Sub CheckParentField()  
  
    Dim pvtTable As PivotTable  
    Dim pvtField As PivotField  
  
    Set pvtTable = ActiveSheet.PivotTables(1)  
    Set pvtField = pvtTable.PivotFields(4)  
  
    ' Check for member properties and notify user.  
    If pvtField.IsMemberProperty = False Then  
        MsgBox "No member properties present."  
    Else  
        MsgBox "The parent field of the members is: " & _  
            pvtField.PropertyParentField  
    End If  
  
End Sub
```



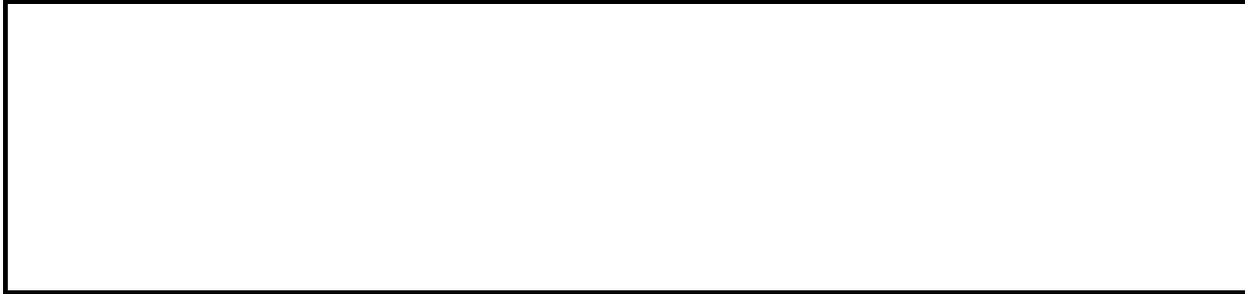
# ProtectChartObject Property

**True** if the embedded chart frame cannot be moved, resized, or deleted.  
Read/write **Boolean**.

## Example

This example protects embedded chart one on worksheet one.

```
Worksheets(1).ChartObjects(1).ProtectChartObject = True
```



# ProtectContents Property

**True** if the contents of the sheet are protected. For a chart, this protects the entire chart. For a worksheet, this protects the individual cells. Read-only **Boolean**.

## Example

This example displays a message box if the contents of Sheet1 are protected.

```
If Worksheets("Sheet1").ProtectContents = True Then  
    MsgBox "The contents of Sheet1 are protected."  
End If
```



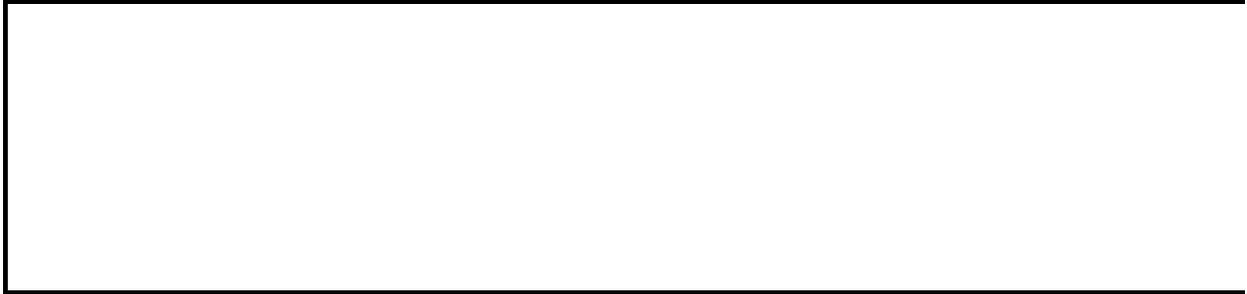
# ProtectData Property

**True** if series formulas cannot be modified by the user. Read/write **Boolean**.

## Example

This example protects the data on embedded chart one on worksheet one.

```
Worksheets(1).ChartObjects(1).Chart.ProtectData = True
```



# ProtectDrawingObjects Property

**True** if shapes are protected. Read-only **Boolean**.

## Example

This example displays a message box if the shapes on Sheet1 are protected.

```
If Worksheets("Sheet1").ProtectDrawingObjects = True Then  
    MsgBox "The shapes on Sheet1 are protected."  
End If
```



# ProtectFormatting Property

**True** if chart formatting cannot be modified by the user. Read/write **Boolean**.

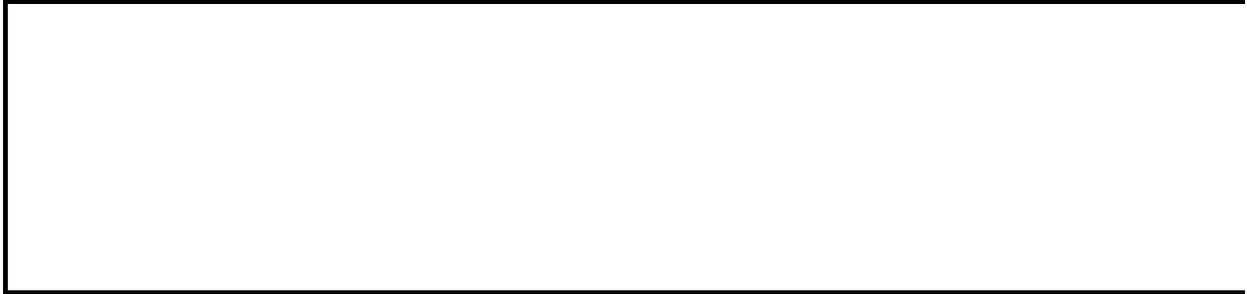
## Remarks

When this property is **True**, the **Object** command on the **Format** menu is disabled and chart elements cannot be added, moved, resized, or deleted.

## Example

This example protects the formatting of embedded chart one on worksheet one.

```
Worksheets(1).ChartObjects(1).Chart.ProtectFormatting = True
```



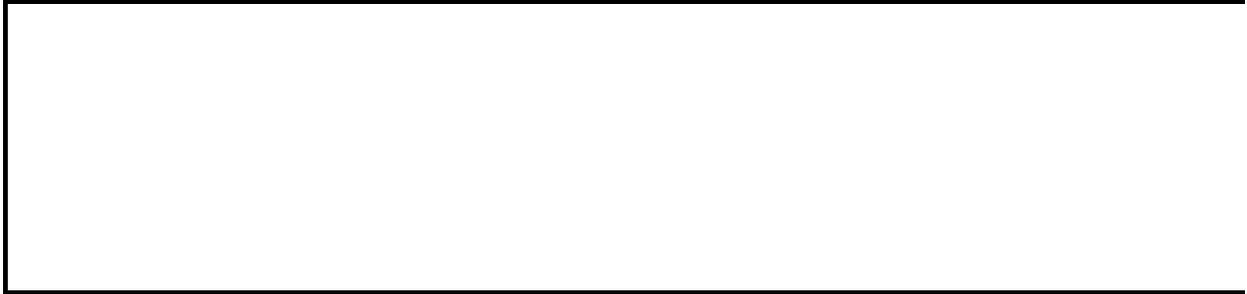
# ProtectGoalSeek Property

**True** if the user cannot modify chart data points with mouse actions. Read/write **Boolean**.

## Example

This example protects the data points on embedded chart one on worksheet one.

```
Worksheets(1).ChartObjects(1).Chart.ProtectGoalSeek = True
```



# Protection Property

Returns a [Protection](#) object that represents the protection options of the worksheet.

*expression*.**Protection**

*expression* Required. An expression that returns a [Worksheet](#) object.

## Example

This example protects the active worksheet and then determines if columns can be inserted on the protected worksheet, notifying the user of this status.

```
Sub CheckProtection()  
    ActiveSheet.Protect  
  
    ' Check the ability to insert columns on a protected sheet.  
    ' Notify the user of this status.  
    If ActiveSheet.Protection.AllowInsertingColumns = True Then  
        MsgBox "The insertion of columns is allowed on this protecte  
    Else  
        MsgBox "The insertion of columns is not allowed on this prot  
    End If  
  
End Sub
```



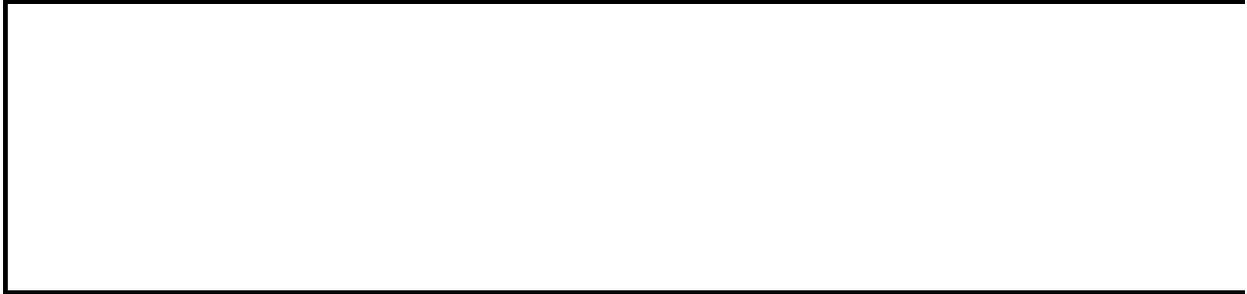
# ProtectionMode Property

**True** if user-interface-only protection is turned on. To turn on user interface protection, use the [Protect](#) method with the *UserInterfaceOnly* argument set to **True**. Read-only **Boolean**.

## Example

This example displays the status of the **ProtectionMode** property.

```
MsgBox ActiveSheet.ProtectionMode
```



# ProtectScenarios Property

**True** if the worksheet scenarios are protected. Read-only **Boolean**.

## Example

This example displays a message box if scenarios are protected on Sheet1.

```
If Worksheets("Sheet1").ProtectScenarios Then _  
    MsgBox "Scenarios are protected on this worksheet."
```



# ProtectSelection Property

**True** if chart elements cannot be selected. Read/write **Boolean**.

## Remarks

When this property is **True**, shapes cannot be added to the chart, and the Click and DoubleClick events for chart elements don't occur.

## Example

This example prevents chart elements from being selected on embedded chart one on worksheet one.

```
Worksheets(1).ChartObjects(1).Chart.ProtectSelection = True
```



# ProtectStructure Property

**True** if the order of the sheets in the workbook is protected. Read-only **Boolean**.

## Example

This example displays a message if the order of the sheets in the active workbook is protected.

```
If ActiveWorkbook.ProtectStructure = True Then
    MsgBox "Remember, you cannot delete, add, or change " & _
        Chr(13) & _
        "the location of any sheets in this workbook."
End If
```



# ProtectWindows Property

**True** if the windows of the workbook are protected. Read-only **Boolean**.

## Example

This example displays a message if the windows in the active workbook are protected.

```
If ActiveWorkbook.ProtectWindows = True Then  
    MsgBox "Remember, you cannot rearrange any" & _  
        " window in this workbook."  
End If
```



# PublishObjects Property

Returns the [PublishObjects](#) collection. Read-only.

## Example

This example publishes all static **PublishObject** objects in the active workbook to the Web page.

```
Set objPObj = ActiveWorkbook.PublishObjects
For Each objPO in objPObj
    If objPO.HtmlType = xlHTMLStatic Then
        objPO.Publish
    End If
Next objPO
```



[Show All](#)

# QueryTable Property

 [QueryTable property as it applies to the \*\*ListObject\*\* object.](#)

Returns the **QueryTable** object that provides a link for the **ListObject** object to the list server. Read-only.

 [QueryTable property as it applies to the \*\*Range\*\* object.](#)

Returns a **QueryTable** object that represents the query table that intersects the specified **Range** object. Read-only.

## Example

This example refreshes the **QueryTable** object that intersects cell A10 on worksheet one.

```
Worksheets(1).Range("a10").QueryTable.Refresh
```

The following example creates a connection to a SharePoint site and publishes the **ListObject** object named List1 to the server. A reference to the **QueryTable** object for the list object is created and the [MaintainConnection](#) property of the **QueryTable** object is set to **True** so that the connection to the SharePoint site is maintained between trips to the server.

```
Dim wrksht As Worksheet
Dim objListObj As ListObject
Dim objQryTbl As QueryTable
Dim prpQryProp As pro
Dim arTarget(4) As String
Dim strSTSCONNECTION As String

Set wrksht = ActiveWorkbook.Worksheets("Sheet1")
Set objListObj = wrksht.ListObjects(1)

arTarget(0) = "0"
arTarget(1) = "http://myteam/project1"
arTarget(2) = "1"
arTarget(3) = "List1"

strSTSCONNECTION = objListObj.Publish(arTarget, True)

Set objQryTbl = objListObj.QueryTable

objQryTbl.MaintainConnection = True
```

# QueryTables Property

Returns the [QueryTables](#) collection that represents all the query tables on the specified worksheet. Read-only.

For more information about returning a single object from a collection, see [Returning an Object from a Collection](#).

## Example

This example refreshes all query tables on worksheet one.

```
For Each qt in Worksheets(1).QueryTables  
    qt.Refresh  
Next
```

This example sets query table one so that formulas to the right of it are automatically updated whenever it's refreshed.

```
Sheets("sheet1").QueryTables(1).FillAdjacentFormulas = True
```



[Show All](#)

# QueryType Property

Indicates the type of query used by Microsoft Excel to populate the query table or PivotTable cache. Read-only [XlQueryType](#).

XlQueryType can be one of these XlQueryType constants.

**xlTextImport**. Based on a text file, for query tables only

**xlOLEDBQuery**. Based on an OLE DB query, including [OLAP](#) data sources

**xlWebQuery**. Based on a Web page, for query tables only

**xlADOREcordset**. Based on an ADO recordset query

**xlDAORecordSet**. Based on a DAO recordset query, for query tables only

**xlODBCQuery**. Based on an ODBC data source

*expression*. **QueryType**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

You specify the data source in the prefix for the [Connection](#) property's value.

## Example

This example refreshes the first query table on the first worksheet if the table is based on a Web page.

```
Set qtQtrResults = _  
    Workbooks(1).Worksheets(1).QueryTables(1)  
With qtQtrResults  
    if .QueryType = xlWebQuery Then  
        .Refresh  
    End If  
End With
```



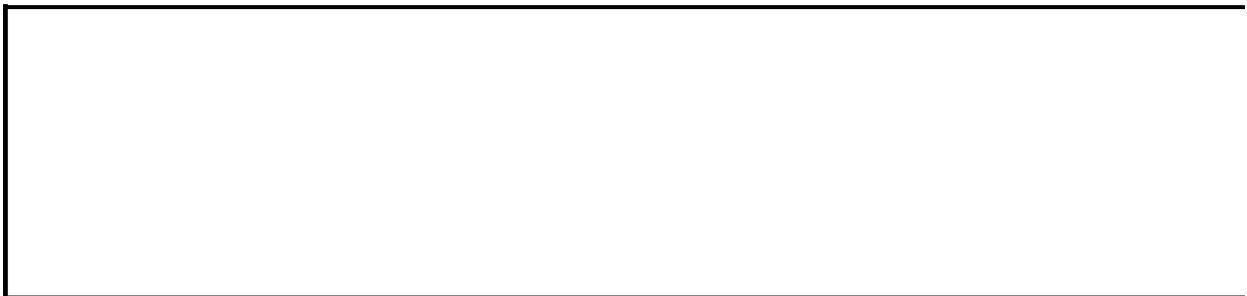
# RadarAxisLabels Property

Returns a [TickLabels](#) object that represents the radar axis labels for the specified chart group. Read-only.

## Example

This example turns on radar axis labels for chart group one in Chart1 and then sets the color for the labels. The example should be run on a radar chart.

```
With Charts("Chart1").ChartGroups(1)  
    .HasRadarAxisLabels = True  
    .RadarAxisLabels.Font.ColorIndex = 3  
End With
```



# RadioGroupSelection Property

Sets or returns a **Long** that represents the index number of the selected item in a group of radio button controls in a smart document. Read/write.

*expression*.**RadioGroupSelection**

*expression* Required. An expression that returns a [SmartTagAction](#) object.

## Remarks

For more information on smart documents, please see the Smart Document Software Development Kit on the [Microsoft Developer Network \(MSDN\)](#) Web site.

---

[Show All](#)

# Range Property

 [Range property as it applies to the \*\*AllowEditRange\*\* object.](#)

Returns a **Range** object that represents a subset of the ranges that can be edited edited on a protected worksheet.

*expression*.**Range**

*expression* Required. An expression that returns an **AllowEditRange** object.

 [Range property as it applies to the \*\*Application\*\*, \*\*Range\*\*, and \*\*Worksheet\*\* objects.](#)

Returns a **Range** object that represents a cell or a range of cells.

*expression*.**Range**(*Cell1*, *Cell2*)

*expression* Required. An expression that returns one of the above objects.

**Cell1** Required **Variant**. The name of the range. This must be an A1-style reference in the language of the macro. It can include the range operator (a colon), the intersection operator (a space), or the union operator (a comma). It can also include dollar signs, but they're ignored. You can use a local defined name in any part of the range. If you use a name, the name is assumed to be in the language of the macro.

**Cell2** Optional **Variant**. The cell in the upper-left and lower-right corner of the range. Can be a **Range** object that contains a single cell, an entire column, or entire row, or it can be a string that names a single cell in the language of the macro.

## Remarks

When used without an object qualifier, this property is a shortcut for `ActiveSheet.Range` (it returns a range from the active sheet; if the active sheet isn't a worksheet, the property fails).

When applied to a **Range** object, the property is relative to the **Range** object. For example, if the selection is cell C3, then `Selection.Range("B1")` returns cell D3 because it's relative to the **Range** object returned by the **Selection** property. On the other hand, the code `ActiveSheet.Range("B1")` always returns cell B1.

 [Range property as it applies to the ListColumn, ListObject, ListRow, HeaderRowRange, InsertRowRange, and TotalsRowRange objects.](#)

Returns a **Range** object that represents the range to which the specified list object in the above list applies. Read-Only.

*expression*.**Range**

*expression* Required. An expression that returns one of the above objects.

 [Range property as it applies to the AutoFilter, Hyperlink, PivotCell, and SmartTag objects.](#)

For an **AutoFilter** object, returns a **Range** object that represents the range to which the specified AutoFilter applies. For a **Hyperlink** object, returns a **Range** object that represents the range the specified hyperlink is attached to. For a **PivotCell** object, returns a **Range** object that represents the range the specified PivotCell applies to. For a **SmartTag** object, returns a **Range** object that represents the range the specified smart tag applies to.

*expression*.**Range**

*expression* Required. An expression that returns one of the above objects.

 [Range property as it applies to the AllowEditRange object.](#)

Returns a [Range](#) object that represents a subset of the ranges that can be edited edited on a protected worksheet.

*expression*.**Range**

*expression* Required. An expression that returns an **AllowEditRange** object.



[Range property as it applies to the GroupShapes and Shapes objects.](#)

Returns a [ShapeRange](#) object that represents a subset of the shapes in a **Shapes** collection.

*expression*.**Range**(*Index*)

*expression* Required. An expression that returns one of the above objects.

**Index** Required **Variant**. The individual shapes to be included in the range. Can be an integer that specifies the index number of the shape, a string that specifies the name of the shape, or an array that contains either integers or strings.

## Remarks

Although you can use the [Range](#) property to return any number of shapes, it's simpler to use the [Item](#) method if you only want to return a single member of the collection. For example, `Shapes(1)` is simpler than `Shapes.Range(1)`.

To specify an array of integers or strings for ***Index***, you can use the **Array** function. For example, the following instruction returns two shapes specified by name.

```
Dim arShapes() As Variant
Dim objRange As Object
arShapes = Array("Oval 4", "Rectangle 5")
Set objRange = ActiveSheet.Shapes.Range(arShapes)
```

In Microsoft Excel, you cannot use this property to return a **ShapeRange** object containing all the **Shape** objects on a worksheet. Instead, use the following code:

```
Worksheets(1).Shapes.Select      ' select all shapes
set sr = Selection.ShapeRange    ' create ShapeRange
```

## Example

[As it applies to the \*\*Application\*\*, \*\*Range\*\*, and \*\*Worksheet\*\* objects.](#)

This example sets the value of cell A1 on Sheet1 to 3.14159.

```
Worksheets("Sheet1").Range("A1").Value = 3.14159
```

This example creates a formula in cell A1 on Sheet1.

```
Worksheets("Sheet1").Range("A1").Formula = "=10*RAND()"
```

This example loops on cells A1:D10 on Sheet1. If one of the cells has a value less than 0.001, the code replaces that value with 0 (zero).

```
For Each c in Worksheets("Sheet1").Range("A1:D10")  
    If c.Value < .001 Then  
        c.Value = 0  
    End If  
Next c
```

This example loops on the range named "TestRange" and displays the number of empty cells in the range.

```
numBlanks = 0  
For Each c In Range("TestRange")  
    If c.Value = "" Then  
        numBlanks = numBlanks + 1  
    End If  
Next c  
MsgBox "There are " & numBlanks & " empty cells in this range"
```

This example sets the font style in cells A1:C5 on Sheet1 to italic. The example uses Syntax 2 of the **Range** property.

```
Worksheets("Sheet1").Range(Cells(1, 1), Cells(5, 3)).  
    Font.Italic = True
```

[As it applies to the \*\*AutoFilter\*\*, \*\*Hyperlink\*\*, \*\*PivotCell\*\*, and \*\*SmartTag\*\* objects.](#)

The following example stores in a variable the address for the AutoFilter applied to the Crew worksheet.

```
rAddress = Worksheets("Crew").AutoFilter.Range.Address
```

This example scrolls through the workbook window until the hyperlink range is in the upper-left corner of the active window.

```
Workbooks(1).Activate  
Set hr = ActiveSheet.Hyperlinks(1).Range  
ActiveWindow.ScrollRow = hr.Row  
ActiveWindow.ScrollColumn = hr.Column
```

[As it applies to the \*\*GroupShapes\*\* and \*\*Shapes\*\* objects.](#)

This example sets the fill pattern for shapes one and three on myDocument.

```
Set myDocument = Worksheets(1)  
myDocument.Shapes.Range(Array(1, 3)) _  
    .Fill.Patterned msoPatternHorizontalBrick
```

This example sets the fill pattern for the shapes named "Oval 4" and "Rectangle 5" on myDocument.

```
Dim arShapes() As Variant  
Dim objRange As Object  
Set myDocument = Worksheets(1)  
arShapes = Array("Oval 4", "Rectangle 5")  
Set objRange = myDocument.Shapes.Range(arShapes)  
objRange.Fill.Patterned msoPatternHorizontalBrick
```

This example sets the fill pattern for shape one on myDocument.

```
Set myDocument = Worksheets(1)  
Set myRange = myDocument.Shapes.Range(1)  
myRange.Fill.Patterned msoPatternHorizontalBrick
```

This example creates an array that contains all the AutoShapes on myDocument, uses that array to define a shape range, and then distributes all the shapes in that range horizontally.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes
    numShapes = .Count
    If numShapes > 1 Then
        numAutoShapes = 1
        ReDim autoShpArray(1 To numShapes)
        For i = 1 To numShapes
            If .Item(i).Type = msoAutoShape Then
                autoShpArray(numAutoShapes) = .Item(i).Name
                numAutoShapes = numAutoShapes + 1
            End If
        Next
        If numAutoShapes > 1 Then
            ReDim Preserve autoShpArray(1 To numAutoShapes)
            Set asRange = .Range(autoShpArray)
            asRange.Distribute msoDistributeHorizontally, False
        End If
    End If
End With
```



# RangeSelection Property

Returns a [Range](#) object that represents the selected cells on the worksheet in the specified window even if a graphic object is active or selected on the worksheet. Read-only.

## Remarks

When a graphic object is selected on a worksheet, the **Selection** property returns the graphic object instead of a **Range** object; the **RangeSelection** property returns the range of cells that was selected before the graphic object was selected.

This property and the **Selection** property return identical values when a range (not a graphic object) is selected on the worksheet.

If the active sheet in the specified window isn't a worksheet, this property fails.

## Example

This example displays the address of the selected cells on the worksheet in the active window.

```
MsgBox ActiveWindow.RangeSelection.Address
```



# ReadingOrder Property

Returns or sets the reading order for the specified object. Can be one of the following constants: **xIRTL** (right-to-left), **xLTR** (left-to-right), or **xContext**.  
Read/write **Long**.

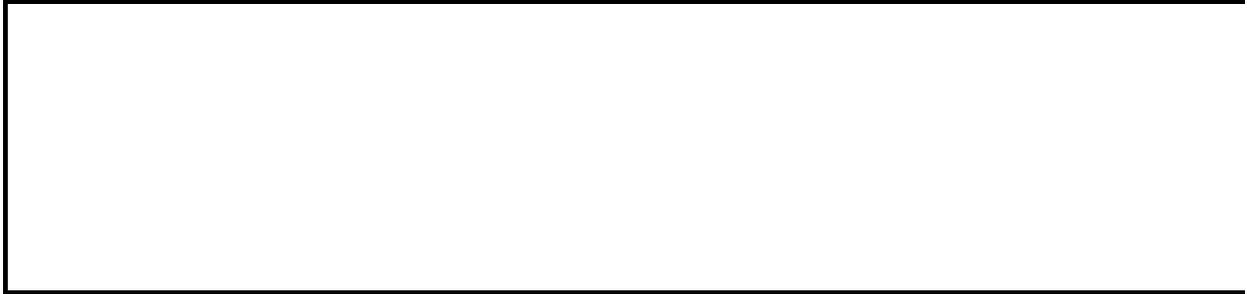
## Remarks

Some of these constants may not be available to you, depending on the language support (U.S. English, for example) that you've selected or installed.

## Example

This example sets the reading order to right-to-left for the chart title of Chart1.

```
Charts("Chart1").ChartTitle.ReadingOrder = x1RTL
```



# ReadOnly Property

Returns **True** if the object has been opened as read-only. Read-only **Boolean**.

This property is used only for lists that are linked to a SharePoint site.

*expression*.**ReadOnly**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

If the active workbook is read-only, this example saves it as Newfile.xls.

```
If ActiveWorkbook.ReadOnly Then
    ActiveWorkbook.SaveAs fileName:="NEWFILE.XLS"
End If
```

The following example displays the setting of the **ReadOnly** property for the third column of a list in Sheet1 of the active workbook.

```
Dim wrksht As Worksheet
Dim objListCol As ListColumn

Set wrksht = ActiveWorkbook.Worksheets("Sheet1")
Set objListCol = wrksht.ListObjects(1).ListColumns(3)

Debug.Print objListCol.ListDataFormat.ReadOnly
```



# ReadOnlyRecommended Property

**True** if the workbook was saved as read-only recommended. Read-only  
**Boolean.**

## Remarks

When you open a workbook that was saved as read-only recommended, Microsoft Excel displays a message recommending that you open the workbook as read-only.

Use the [SaveAs](#) method to change this property.

## Example

This example displays a message if the active workbook is saved as read-only recommended.

```
If ActiveWorkbook.ReadOnlyRecommended = True Then  
    MsgBox "This workbook is saved as read-only recommended"  
End If
```



# Ready Property

Returns **True** when the Microsoft Excel application is ready; **False** when the Excel application is not ready. Read-only **Boolean**.

*expression*.**Ready**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

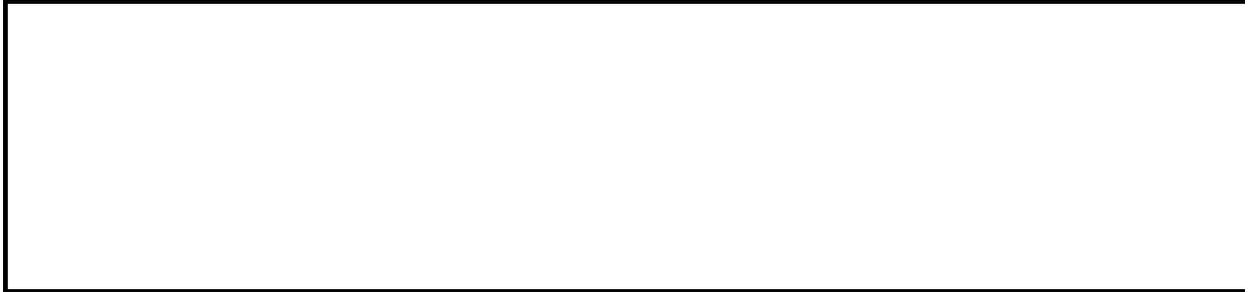
In this example, Microsoft Excel checks to see if the **Ready** property is set to **True**, and if so, a message displays "Application is ready." Otherwise, Excel displays the message "Application is not ready."

```
Sub UseReady()  
    If Application.Ready = True Then  
        MsgBox "Application is ready."  
    Else  
        MsgBox "Application is not ready."  
    End If  
End Sub
```



# Received Property

You have requested Help for a Visual Basic keyword used only on the Macintosh. For information about this keyword, consult the language reference Help included with Microsoft Office Macintosh Edition.



# RecentFiles Property

Returns a [RecentFiles](#) collection that represents the list of recently used files.

For information about returning a single object from a collection, see [Returning an Object from a Collection](#).

## Example

This example sets the maximum number of files in the list of recently used files to 6.

```
Application.RecentFiles.Maximum = 6
```



# Recipients Property

Returns or sets the recipients on the routing slip.

*expression*.**Recipients**(*Index*)

*expression* Required. An expression that returns a **RoutingSlip** object.

*Index* Optional **Variant**. The recipient. If this argument isn't specified, the **Recipients** property returns (or can be set to) an array that contains all recipients.

## Remarks

The order of the recipient list defines the delivery order if the routing delivery option is **xlOneAfterAnother**.

If a routing slip is in progress, only those recipients who haven't already received and routed the document are returned or set.

## Example

This example sends the open workbook to three recipients, one after the other.

```
With ThisWorkbook
    .HasRoutingSlip = True
    With .RoutingSlip
        .Delivery = xlOneAfterAnother
        .Recipients = Array("Adam Bendel", _
            "Jean Selva", "Bernard Gabor")
        .Subject = "Here is the workbook"
        .Message = "Here is the workbook. What do you think?"
        .ReturnWhenDone = True
    End With
    .Route
End With
```



# Recognize Property

Returns **True** when data can be labeled as a smart tag. Read/write **Boolean**.

*expression*.**Recognize**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel determines if the ability to label data as smart tags is enabled and notifies the user.

```
Sub CheckSmartTagRecognition()  
    ' Determine if data can be labeled as SmartTags.  
    If Application.SmartTagRecognizers.Recognize = True Then  
        MsgBox "Background smart tag recognition is turned on."  
    Else  
        MsgBox "Background smart tag recognition is turned off."  
    End If  
End Sub
```



# RecordCount Property

Returns the number of records in the PivotTable cache or the number of cache records that contain the specified item. Read-only **Long**.

## Remarks

This property reflects the transient state of the cache at the time that it's queried. The cache can change between queries.

## Example

This example displays the number of cache records that contain "Kiwi" in the "Products" field.

```
MsgBox Worksheets(1).PivotTables("Pivot1") _  
    .PivotFields("Product").PivotItems("Kiwi").RecordCount
```



# RecordRelative Property

**True** if macros are recorded using relative references; **False** if recording is absolute. Read-only **Boolean**.

## Example

This example displays the address of the active cell on Sheet1 in A1 style if **RecordRelative** is **False**; otherwise, it displays the address in R1C1 style.

```
Worksheets("Sheet1").Activate  
If Application.RecordRelative = False Then  
    MsgBox ActiveCell.Address(ReferenceStyle:=xlA1)  
Else  
    MsgBox ActiveCell.Address(ReferenceStyle:=xlR1C1)  
End If
```



# Recordset Property

Returns or sets a **Recordset** object that's used as the data source for the specified query table or PivotTable cache. Read/write.

## Remarks

If this property is used to overwrite an existing recordset, the change takes effect when the [Refresh](#) method is run.

## Example

This example changes the **Recordset** object used with the first query table on the first worksheet and then refreshes the query table.

```
With Worksheets(1).QueryTables(1)
    .Recordset = _
        OpenDatabase("c:\Nwind.mdb") _
        .OpenRecordset("employees")
    .Refresh
End With
```

This example creates a new PivotTable cache using an ADO connection to Microsoft Jet, and then it creates a new PivotTable report based on the cache, at cell A3 on the active worksheet.

```
Dim cnnConn As ADODB.Connection
Dim rstRecordset As ADODB.Recordset
Dim cmdCommand As ADODB.Command

' Open the connection.
Set cnnConn = New ADODB.Connection
With cnnConn
    .ConnectionString = _
        "Provider=Microsoft.Jet.OLEDB.4.0"
    .Open "C:\perfdate\record.mdb"
End With

' Set the command text.
Set cmdCommand = New ADODB.Command
Set cmdCommand.ActiveConnection = cnnConn
With cmdCommand
    .CommandText = "Select Speed, Pressure, Time From DynoRun"
    .CommandType = adCmdText
    .Execute
End With

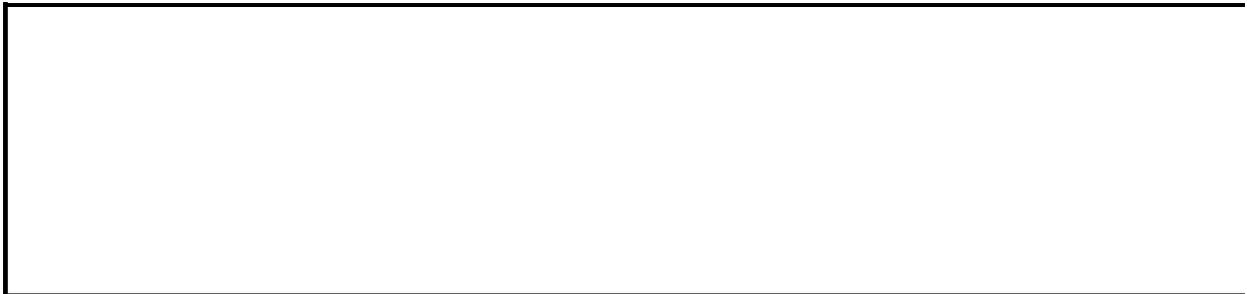
' Open the recordset.
Set rstRecordset = New ADODB.Recordset
Set rstRecordset.ActiveConnection = cnnConn
rstRecordset.Open cmdCommand

' Create a PivotTable cache and report.
Set objPivotCache = ActiveWorkbook.PivotCaches.Add( _
```

```
        SourceType:=xlExternal)
Set objPivotCache.Recordset = rstRecordset
With objPivotCache
    .CreatePivotTable TableDestination:=Range("A3"), _
        TableName:="Performance"
End With

With ActiveSheet.PivotTables("Performance")
    .SmallGrid = False
    With .PivotFields("Pressure")
        .Orientation = xlRowField
        .Position = 1
    End With
    With .PivotFields("Speed")
        .Orientation = xlColumnField
        .Position = 1
    End With
    With .PivotFields("Time")
        .Orientation = xlDataField
        .Position = 1
    End With
End With

' Close the connections and clean up.
cnnConn.Close
Set cmdCommand = Nothing
Set rstRecordSet = Nothing
Set cnnConn = Nothing
```



[Show All](#)

# ReferenceStyle Property

Returns or sets how Microsoft Excel displays cell references and row and column headings in either A1 or R1C1 reference style. Read/write [XlReferenceStyle](#).

XlReferenceStyle can be one of these XlReferenceStyle constants.

**xlA1**

**xlR1C1**

*expression*.**ReferenceStyle**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example displays the current reference style.

```
If Application.ReferenceStyle = xlR1C1 Then
    MsgBox ("Microsoft Excel is using R1C1 references")
Else
    MsgBox ("Microsoft Excel is using A1 references")
End If
```



# RefersTo Property

Returns or sets the formula that the name is defined to refer to, in the language of the macro and in A1-style notation, beginning with an equal sign. Read/write **String**.

## Example

This example creates a list of all the names in the active workbook, and it shows their formulas in A1-style notation in the language of the macro. The list appears on a new worksheet created by the example.

```
Set newSheet = Worksheets.Add
i = 1
For Each nm In ActiveWorkbook.Names
    newSheet.Cells(i, 1).Value = nm.Name
    newSheet.Cells(i, 2).Value = "'" & nm.Refersto
    i = i + 1
Next
newSheet.Columns("A:B").AutoFit
```



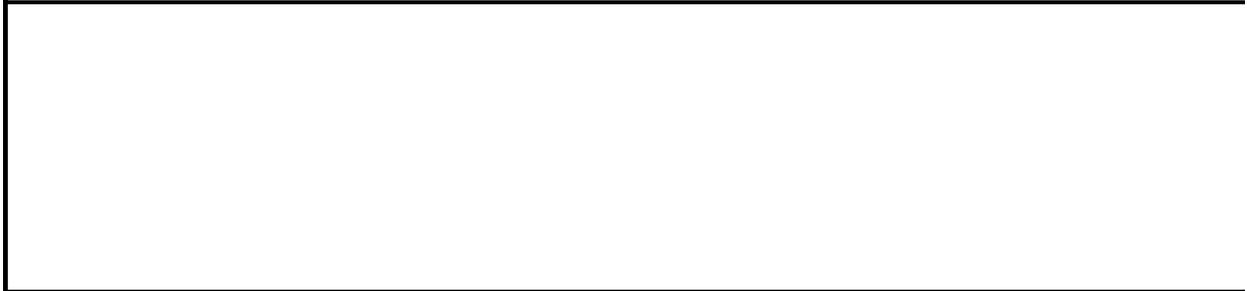
# RefersToLocal Property

Returns or sets the formula that the name refers to. The formula is in the language of the user, and it's in A1-style notation, beginning with an equal sign. Read/write **String**.

## Example

This example creates a new worksheet and then inserts a list of all the names in the active workbook, including their formulas (in A1-style notation and in the language of the user).

```
Set newSheet = ActiveWorkbook.Worksheets.Add
i = 1
For Each nm In ActiveWorkbook.Names
    newSheet.Cells(i, 1).Value = nm.NameLocal
    newSheet.Cells(i, 2).Value = "'" & nm.ReferstoLocal
    i = i + 1
Next
```



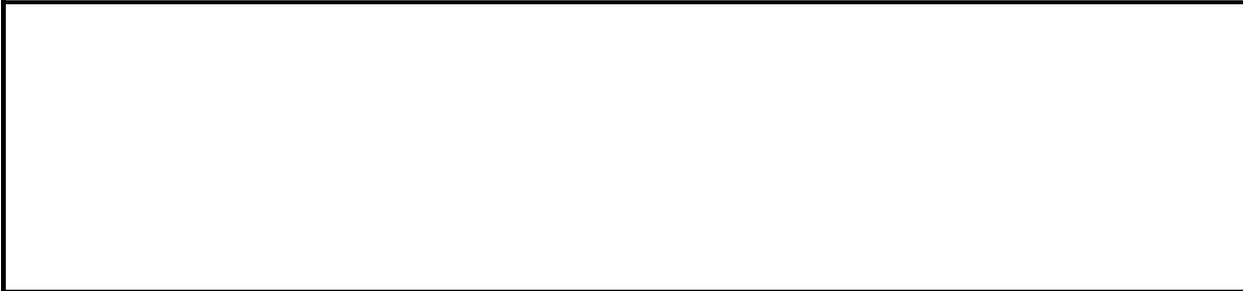
# RefersToR1C1 Property

Returns or sets the formula that the name refers to. The formula is in the language of the macro, and it's in R1C1-style notation, beginning with an equal sign. Read/write **String**.

## Example

This example creates a new worksheet and then inserts a list of all the names in the active workbook, including their formulas (in R1C1-style notation and in the language of the macro).

```
Set newSheet = ActiveWorkbook.Worksheets.Add
i = 1
For Each nm In ActiveWorkbook.Names
    newSheet.Cells(i, 1).Value = nm.Name
    newSheet.Cells(i, 2).Value = "'" & nm.ReferstoR1C1
    i = i + 1
Next
```



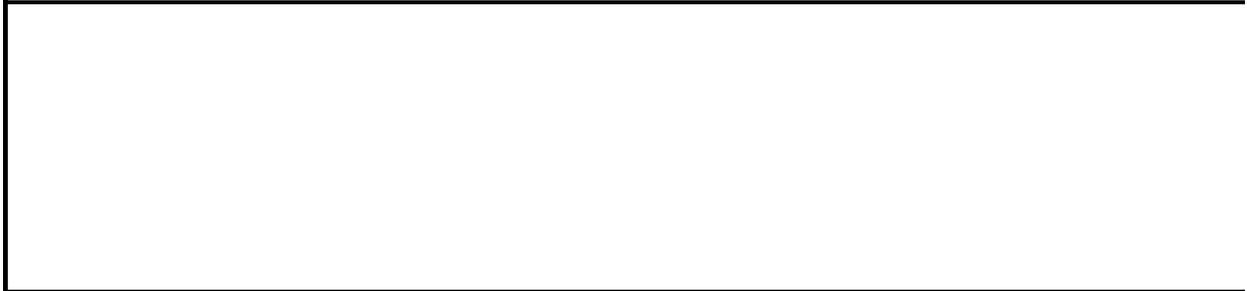
# RefersToR1C1Local Property

Returns or sets the formula that the name refers to. This formula is in the language of the user, and it's in R1C1-style notation, beginning with an equal sign. Read/write **String**.

## Example

This example creates a new worksheet and then inserts a list of all the names in the active workbook, including their formulas (in R1C1-style notation and in the language of the user).

```
Set newSheet = ActiveWorkbook.Worksheets.Add
i = 1
For Each nm In ActiveWorkbook.Names
    newSheet.Cells(i, 1).Value = nm.NameLocal
    newSheet.Cells(i, 2).Value = "'" & nm.ReferstoR1C1Local
    i = i + 1
Next
```



# RefersToRange Property

Returns the **Range** object referred to by a **Name** object. Read-only.

## Remarks

If the **Name** object doesn't refer to a range (for example, if it refers to a constant or a formula), this property fails.

To change the range that a name refers to, use the [RefersTo](#) property.

## Example

This example displays the number of rows and columns in the print area on the active worksheet.

```
p = Names("Print_Area").ReferstoRange.Value  
MsgBox "Print_Area: " & UBound(p, 1) & " rows, " & _  
    UBound(p, 2) & " columns"
```



[Show All](#)

# RefreshDate Property

Returns the date on which the PivotTable report or cache was last refreshed.  
Read-only **Date**.

## Remarks

For **PivotCache** objects, the cache must have at least one PivotTable report associated with it.

For [OLAP](#) data sources, this property is updated after each query.

## Example

This example displays the date on which the PivotTable report was last refreshed.

```
Set pvtTable = Worksheets("Sheet1").Range("A3").PivotTable
dateString = Format(pvtTable.RefreshDate, "Long Date")
MsgBox "The data was last refreshed on " & dateString
```



# Refreshing Property

**True** if there's a background query in progress for the specified query table.  
Read/write **Boolean**.

## Remarks

Use the [CancelRefresh](#) method to cancel background queries.

## Example

This example displays a message box if there's a background query in progress for query table one.

```
With Worksheets(1).QueryTables(1)
  If .Refreshing Then
    MsgBox "Query is currently refreshing: please wait"
  Else
    .Refresh BackgroundQuery := False
    .ResultRange.Select
  End If
End With
```



[Show All](#)

# RefreshName Property

Returns the name of the person who last refreshed the PivotTable report data or the PivotTable cache. Read-only **String**.

## Remarks

For [OLAP](#) data sources, this property is updated after each query.

## Example

This example displays the name of the person who last refreshed the PivotTable report.

```
Set pvtTable = Worksheets("Sheet1").Range("A3").PivotTable  
MsgBox "The data was last refreshed by " & pvtTable.RefreshName
```



# RefreshOnChange Property

**True** if the specified query table is refreshed whenever you change the parameter value of a parameter query. Read/write **Boolean**.

## Remarks

You can set this property to **True** only if you use parameters of type **xlRange** and if the referenced parameter value is in a single cell. The refresh occurs when you change the value of the cell.

## Example

This example changes the SQL statement for the first query table on Sheet1. The clause "(ContactTitle=?)" indicates that the query is a parameter query, and the value of the title is set to the value of cell D4. The query table will be automatically refreshed whenever the value of this cell changes.

```
Set objQT = Worksheets("Sheet1").QueryTables(1)
objQT.CommandText = "Select * From Customers Where (ContactTitle=?)"
Set objParam1 = objQT.Parameters _
    .Add("Contact Title", xlParamTypeVarChar)
objParam1.RefreshOnChange = True
objParam1.SetParam xlRange, Range("D4")
```



# RefreshOnFileOpen Property

**True** if the PivotTable cache or query table is automatically updated each time the workbook is opened. The default value is **False**. Read/write **Boolean**.

## Remarks

Query tables and PivotTable reports are not automatically refreshed when you open the workbook by using the **Open** method in Visual Basic. Use the [Refresh](#) method to refresh the data after the workbook is open.

## Example

This example causes the PivotTable cache to automatically update each time the workbook is opened.

```
ActiveWorkbook.PivotCaches(1).RefreshOnFileOpen = True
```



# RefreshPeriod Property

Returns or sets the number of minutes between refreshes. Read/write **Long**.

## Remarks

Setting the period to 0 (zero) disables automatic timed refreshes and is equivalent to setting this property to **Null**.

The value of the **RefreshPeriod** property can be an integer from 0 through 32767.

## Example

This example sets the refresh period for the PivotTable cache (PivotTable3) to 15 minutes.

```
Set objPC = Worksheets("Sheet1").PivotTables("PivotTable3").PivotCache
objPC.RefreshPeriod = 15
```



[Show All](#)

# RefreshStyle Property

Returns or sets the way rows on the specified worksheet are added or deleted to accommodate the number of rows in a recordset returned by a query. Read/write [XlCellInsertionMode](#).

XlCellInsertionMode can be one of these XlCellInsertionMode constants.

**xlInsertDeleteCells.** Partial rows are inserted or deleted to match the exact number of rows required for the new recordset.

**xlOverwriteCells.** No new cells or rows are added to the worksheet. Data in surrounding cells is overwritten to accommodate any overflow.

**xlInsertEntireRows.** Entire rows are inserted, if necessary, to accommodate any overflow. No cells or rows are deleted from the worksheet.

*expression*.**RefreshStyle**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example adds a query table to Sheet1. The **RefreshStyle** property adds rows to the worksheet as needed, to hold the data results.

```
Dim qt As QueryTable
Set qt = Sheets("sheet1").QueryTables _
    .Add(Connection:="Finder;c:\myfile.dqy", _
        Destination:=Range("sheet1!a1"))
With qt
    .RefreshStyle = xlInsertEntireRows
    .Refresh
End With
```



# RegisteredFunctions Property

Returns information about functions in either dynamic-link libraries (DLLs) or code resources that were registered with the REGISTER or REGISTER.ID macro functions. Read-only **Variant**.

*expression*.**RegisteredFunctions**(*Index1*, *Index2*)

*expression* Required. An expression that returns an **Application** object.

*Index1* Optional **Variant**. The name of the DLL or code resource.

*Index2* Optional **Variant**. The name of the function.

## Remarks

If you don't specify the index arguments, this property returns an array that contains a list of all registered functions. Each row in the array contains information about a single function, as shown in the following table.

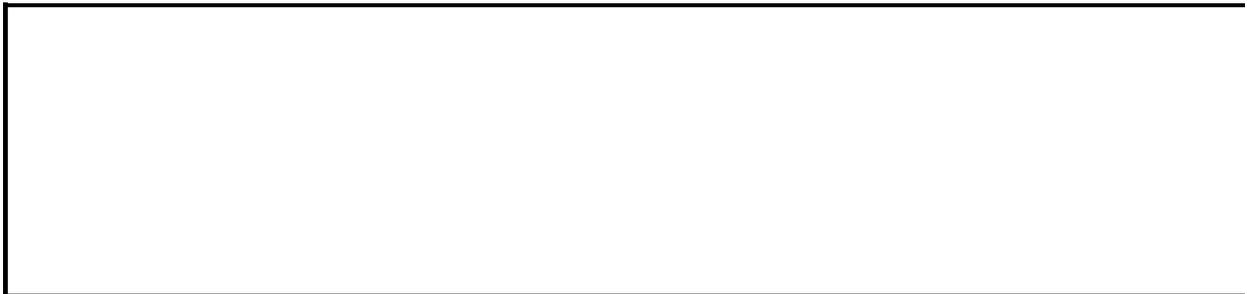
<b>Column</b>	<b>Contents</b>
1	The name of the DLL or code resource
2	The name of the procedure in the DLL or code resource
3	Strings specifying the data types of the return values, and the number and data types of the arguments

If there are no registered functions, this property returns **Null**.

## Example

This example creates a list of registered functions, placing one registered function in each row on Sheet1. Column A contains the full path and file name of the DLL or code resource, column B contains the function name, and column C contains the argument data type code.

```
theArray = Application.RegisteredFunctions
If IsNull(theArray) Then
    MsgBox "No registered functions"
Else
    For i = LBound(theArray) To UBound(theArray)
        For j = 1 To 3
            Worksheets("Sheet1").Cells(i, j). _
                Formula = theArray(i, j)
        Next j
    Next i
End If
```



# RelyOnCSS Property

**True** if cascading style sheets (CSS) are used for font formatting when you view a saved document in a Web browser. Microsoft Excel creates a cascading style sheet file and saves it either to the specified folder or to the same folder as your Web page, depending on the value of the [OrganizeInFolder](#) property. **False** if HTML <FONT> tags and cascading style sheets are used. The default value is **True**. Read/write **Boolean**.

## Remarks

You should set this property to **True** if your Web browser supports cascading style sheets, as this will give you more precise layout and formatting control on your Web page and make it look more like your document (as it appears in Microsoft Excel).

## Example

This example enables the use of cascading style sheets as the global default for the application.

```
Application.DefaultWebOptions.RelyOnCSS = True
```



# RelyOnVML Property

**True** if image files are not generated from drawing objects when you save a document as a Web page. **False** if images are generated. The default value is **False**. Read/write **Boolean**.

## Remarks

You can reduce file sizes by not generating images for drawing objects, if your Web browser supports Vector Markup Language (VML). For example, Microsoft Internet Explorer 5 supports this feature, and you should set the **RelyOnVML** property to **True** if you are targeting this browser. For browsers that do not support VML, the image will not appear when you view a Web page saved with this property enabled.

For example, you should not generate images if your Web page uses image files that you have generated earlier, and if the location where you save the document is different from the final location of the page on the Web server.

## Example

This example specifies that images are generated when saving the worksheet to a Web page.

```
workbooks(1).WebOptions.RelyOnVML = False
```



# RemovePersonalInformation Property

**True** if personal information can be removed from the specified workbook. The default value is **False**. Read/write **Boolean**.

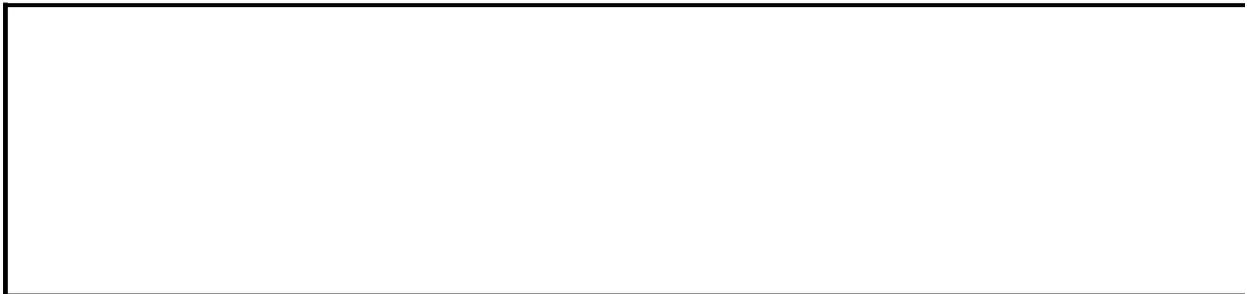
*expression*.**RemovePersonalInformation**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel determines if personal information can be removed from the specified workbook and notifies the user.

```
Sub UsePersonalInformation()  
    Dim wkbOne As Workbook  
  
    Set wkbOne = Application.ActiveWorkbook  
  
    ' Determine settings and notify user.  
    If wkbOne.RemovePersonalInformation = True Then  
        MsgBox "Personal information can be removed."  
    Else  
        MsgBox "Personal information cannot be removed."  
    End If  
  
End Sub
```



# Repeating Property

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Returns **True** if the specified [XPath](#) object is mapped to an XML list; returns **False** if the **XPath** object is mapped to a single cell. Read-only **Boolean**.

*expression.Repeating*

*expression* Required. An expression that returns an **XPath** object.



# RepeatItemsOnEachPrintedPage Property

**True** if row, column, and item labels appear on the first row of each page when the specified PivotTable report is printed. **False** if labels are printed only on the first page. The default value is **True**. Read/write **Boolean**.

## Remarks

The PivotTable report must be the only one in the print area. To set an indented format for a PivotTable report, use the [Format](#) method.

Microsoft Excel prints row and column labels in place of any print titles set for the worksheet. Use the [PrintTitles](#) property to determine whether print titles are set for the PivotTable report.

## Example

This example sets Microsoft Excel to repeat the labels on each page when the fourth PivotTable report on the active worksheet is printed.

```
ActiveSheet.PivotTables("PivotTable4") _  
    .RepeatItemsOnEachPrintedPage = True
```



# ReplaceFormat Property

Sets the replacement criteria to use in replacing cell formats. The replacement criteria is then used in a subsequent call to the Replace method of the Range object.

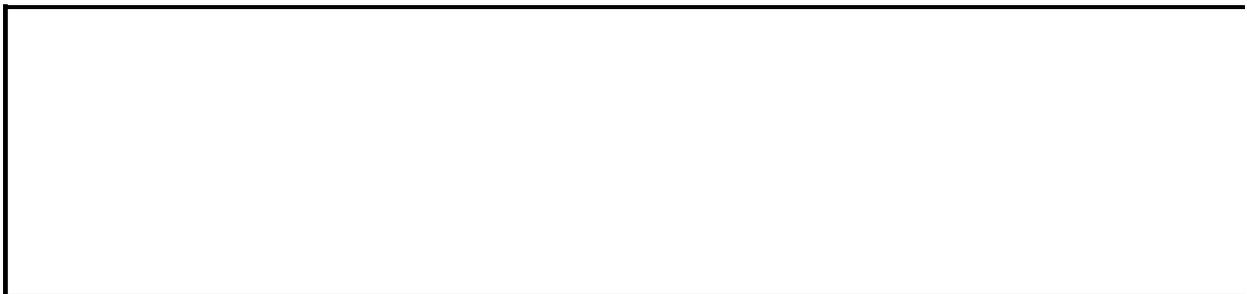
*expression*.**ReplaceFormat**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

The following example sets the search criteria to find cells containing Arial, Regular, Size 10 font, replaces their formats with Arial, Bold, Size 8 font, and then calls the Replace method, with the optional arguments of SearchFormat and ReplaceFormat set to True to actually make the changes.

```
Sub MakeBold()  
  
    ' Establish search criteria.  
    With Application.FindFormat.Font  
        .Name = "Arial"  
        .FontStyle = "Regular"  
        .Size = 10  
    End With  
  
    ' Establish replacement criteria.  
    With Application.ReplaceFormat.Font  
        .Name = "Arial"  
        .FontStyle = "Bold"  
        .Size = 8  
    End With  
  
    ' Notify user.  
    With Application.ReplaceFormat.Font  
        MsgBox .Name & "-" & .FontStyle & "-" & .Size & _  
            " font is what the search criteria will replace cell for  
    End With  
  
    ' Make the replacements in the worksheet.  
    Cells.Replace What:="", Replacement:="", _  
        SearchFormat:=True, ReplaceFormat:=True  
  
End Sub
```



# ReplacementList Property

Returns the array of AutoCorrect replacements.

*expression*.**ReplacementList**(*Index*)

*expression* Required. An expression that returns an **AutoCorrect** object.

**Index** Optional **Variant**. The row index of the array of AutoCorrect replacements to be returned. The row is returned as a one-dimensional array with two elements: The first element is the text in column 1, and the second element is the text in column 2.

## Remarks

If *Index* is not specified, this method returns a two-dimensional array. Each row in the array contains one replacement, as shown in the following table.

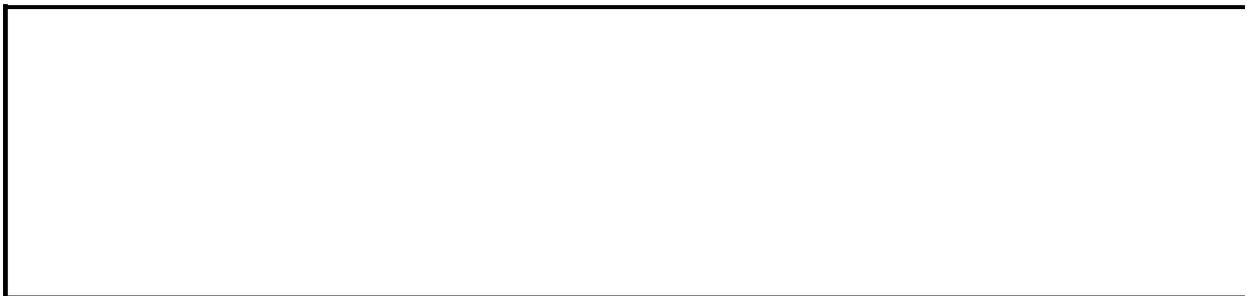
Column	Contents
1	The text to be replaced
2	The replacement text

Use the [AddReplacement](#) method to add an entry to the replacement list.

## Example

This example searches the replacement list for "Temperature" and displays the replacement entry if it exists.

```
repl = Application.AutoCorrect.ReplacementList  
For x = 1 To UBound(repl)  
    If repl(x, 1) = "Temperature" Then MsgBox repl(x, 2)  
Next
```



# ReplaceText Property

**True** if text in the list of AutoCorrect replacements is replaced automatically.  
Read/write **Boolean**.

## Example

This example turns off automatic text replacement.

```
With Application.AutoCorrect  
    .CapitalizeNamesOfDays = True  
    .ReplaceText = False  
End With
```



# Required Property

Returns a **Boolean** value indicating whether the schema definition of a column requires data before the row is committed. Read-only **Boolean**.

This property is used only for lists that are linked to a SharePoint site.

*expression*.**Required**

*expression* Required. An expression that returns a **ListDataFormat** object.

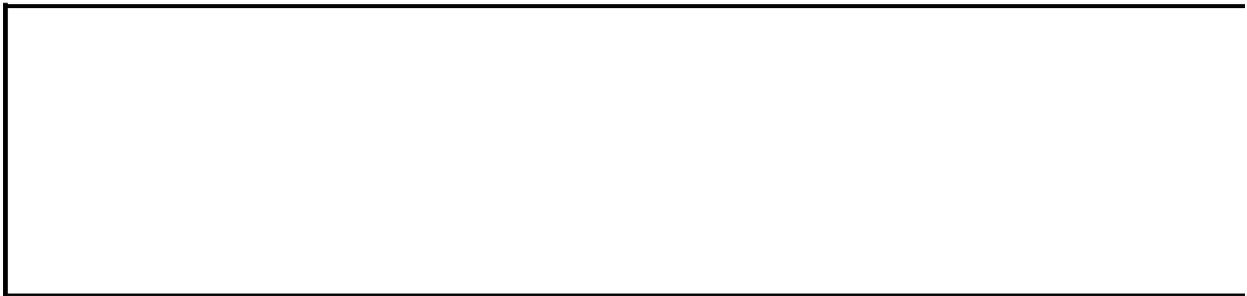
## Remarks

In Microsoft Excel, you cannot set any of the properties associated with the **ListDataFormat** object. You can set these properties, however, by modifying the list on the SharePoint site.

## Example

The following example displays the setting of the **Required** property for the third column of a list in Sheet1 of the active workbook.

```
Sub Test()  
    Dim wrksht As Worksheet  
    Dim objListCol As ListColumn  
  
    Set wrksht = ActiveWorkbook.Worksheets("Sheet1")  
    Set objListCol = wrksht.ListObjects(1).ListColumns(3)  
  
    Debug.Print objListCol.ListDataFormat.Required  
End Sub
```



# Resize Property

Resizes the specified range. Returns a **Range** object that represents the resized range.

*expression*.**Resize**(*RowSize*, *ColumnSize*)

*expression* Required. An expression that returns a **Range** object to be resized.

**RowSize** Optional **Variant**. The number of rows in the new range. If this argument is omitted, the number of rows in the range remains the same.

**ColumnSize** Optional **Variant**. The number of columns in the new range. If this argument is omitted, the number of columns in the range remains the same.

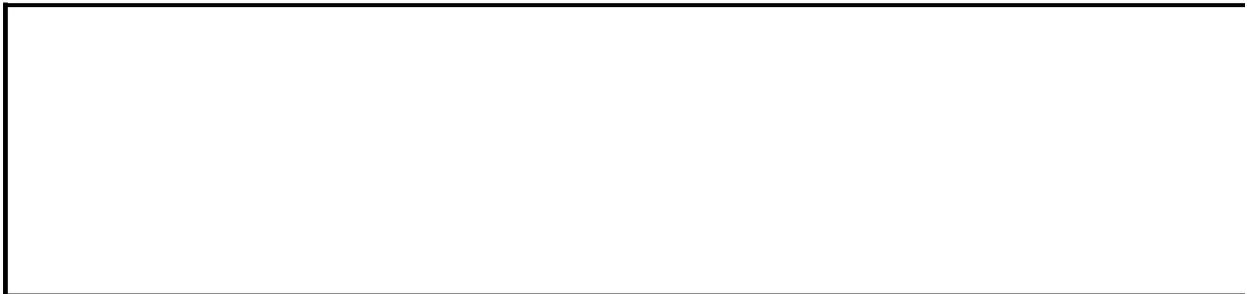
## Example

This example resizes the selection on Sheet1 to extend it by one row and one column.

```
Worksheets("Sheet1").Activate  
numRows = Selection.Rows.Count  
numColumns = Selection.Columns.Count  
Selection.Resize(numRows + 1, numColumns + 1).Select
```

This example assumes that you have a table on Sheet1 that has a header row. The example selects the table, without selecting the header row. The active cell must be somewhere in the table before you run the example.

```
Set tbl = ActiveCell.CurrentRegion  
tbl.Offset(1, 0).Resize(tbl.Rows.Count - 1, _  
tbl.Columns.Count).Select
```



# ResultRange Property

Returns a **Range** object that represents the area of the worksheet occupied by the specified query table. Read-only.

## Remarks

The range doesn't include the field name row or the row number column.

## Example

This example sums the data in the first column of query table one. The sum of the first column is displayed below the data range.

```
Set c1 = Sheets("sheet1").QueryTables(1).ResultRange.Columns(1)
c1.Name = "Column1"
c1.End(xlDown).Offset(2, 0).Formula = "=sum(Column1)"
```



# ReturnWhenDone Property

**True** if the workbook is returned to the sender when routing is finished.  
Read/write **Boolean**.

## Remarks

You cannot set this property if routing is in progress

## Example

This example sends Book1.xls to three recipients, one after another, and then it returns the workbook to the sender when routing has been completed.

```
Workbooks("BOOK1.XLS").HasRoutingSlip = True
With Workbooks("BOOK1.XLS").RoutingSlip
    .Delivery = xlOneAfterAnother
    .Recipients = Array("Adam Bendel", _
        "Jean Selva", "Bernard Gabor")
    .Subject = "Here is BOOK1.XLS"
    .Message = "Here is the workbook. What do you think?"
    .ReturnWhenDone = True
End With
Workbooks("BOOK1.XLS").Route
```



[Show All](#)

# Reverse Property

**MsoTrue** reverses the nodes in a diagram. Read/write [MsoTriState](#).

MsoTriState can be one of these MsoTriState constants.

**msoCTrue** Not used with this property.

**msoFalse** Leaves the diagram nodes as they are.

**msoTriStateMixed** Not used with this property.

**msoTriStateToggle** Not used with this property.

**msoTrue** Reverses the nodes in a diagram.

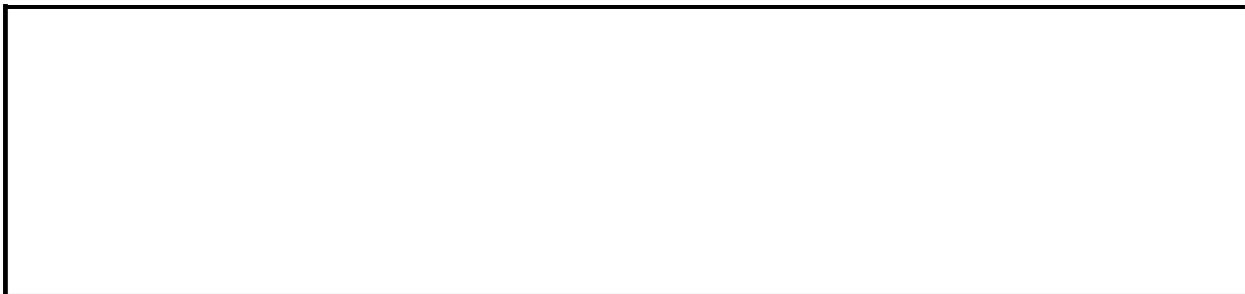
*expression*.**Reverse**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

The following example creates a pyramid diagram, and reverses the nodes so that the node that was on the bottom of the pyramid is on the top, and the node that was on the top is on the bottom.

```
Sub CreatePyramidDiagram()  
  
    Dim shpDiagram As Shape  
    Dim dgnNode As DiagramNode  
    Dim intCount As Integer  
  
    'Add pyramid diagram to the current document  
    Set shpDiagram = ActiveSheet.Shapes.AddDiagram( _  
        Type:=msoDiagramPyramid, Left:=10, _  
        Top:=15, Width:=400, Height:=475)  
  
    'Add first child node to the diagram  
    Set dgnNode = shpDiagram.DiagramNode.Children.AddNode  
  
    'Add three child nodes  
    For intCount = 1 To 3  
        dgnNode.AddNode  
    Next intCount  
  
    With dgnNode.Diagram  
        'Enable automatic formatting  
        .AutoFormat = msoTrue  
  
        'Reverse the order of the nodes  
        .Reverse = msoTrue  
    End With  
  
End Sub
```



# ReversePlotOrder Property

**True** if Microsoft Excel plots data points from last to first. Read/write **Boolean**.

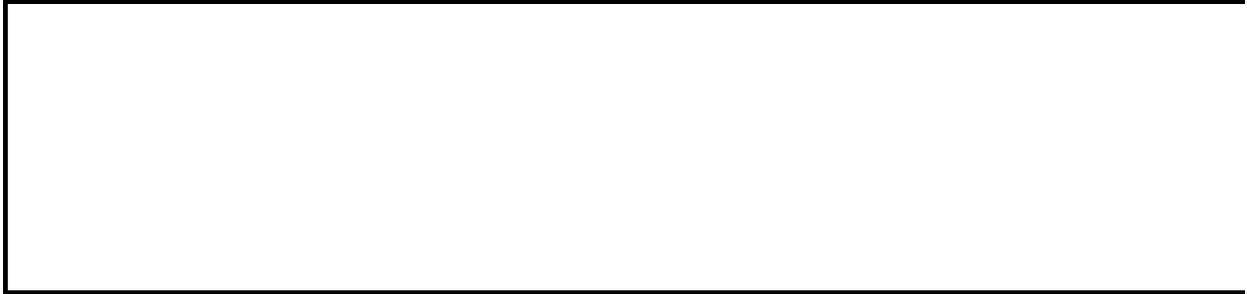
## Remarks

This property cannot be used on radar charts.

## Example

This example plots data points from last to first on the value axis on Chart1.

```
Charts("Chart1").Axes(x1Value).ReversePlotOrder = True
```



# RevisionNumber Property

Returns the number of times the workbook has been saved while open as a shared list. If the workbook is open in exclusive mode, this property returns 0 (zero). Read-only **Long**.

## Remarks

The **RevisionNumber** property is updated only when the local copy of the workbook is saved, not when remote copies are saved.

## Example

This example uses the revision number to determine whether the active workbook is open in exclusive mode. If it is, the example saves the workbook as a shared list.

```
If ActiveWorkbook.RevisionNumber = 0 Then
    ActiveWorkbook.SaveAs _
        filename:=ActiveWorkbook.FullName, _
        accessMode:=xlShared, _
        conflictResolution:= _
            xlOtherSessionChanges
End If
```



[Show All](#)

# RGB Property

 [RGB property as it applies to the \*\*ChartColorFormat\*\* object.](#)

Returns the red-green-blue value of the specified color. Read-only **Long**.

*expression*.**RGB**

*expression* Required. An expression that returns one of the above objects.

 [RGB property as it applies to the \*\*ColorFormat\*\* object.](#)

Returns or sets the red-green-blue value of the specified color. Read/write **Long**.

*expression*.**RGB**

*expression* Required. An expression that returns one of the above objects.

## Example

This example sets the interior color of the range A1:A10 to the chart area foreground fill color on chart one.

```
Worksheets(1).Range("A1:A10").Interior.Color = _  
    Charts(1).ChartArea.Fill.ForeColor.RGB
```



# RightAngleAxes Property

**True** if the chart axes are at right angles, independent of chart rotation or elevation. Applies only to 3-D line, column, and bar charts. Read/write **Boolean**.

## Remarks

If this property is **True**, the [Perspective](#) property is ignored.

## Example

This example sets the axes in Chart1 to intersect at right angles. The example should be run on a 3-D chart.

```
Charts("Chart1").RightAngleAxes = True
```



# RightFooter Property

Returns or sets the right part of the footer. Read/write **String**.

## Remarks

Special [format codes](#) can be used in the footer text.

## Example

This example prints the page number in the lower-right corner of every page.

```
Worksheets("Sheet1").PageSetup.RightFooter = "&P"
```



# RightFooterPicture Property

Returns a **Graphic** object that represents the picture for the right section of the footer. Used to set attributes of the picture.

*expression*.**RightFooterPicture**

*expression* Required. An expression that returns a [PageSetup](#) object.

## Remarks

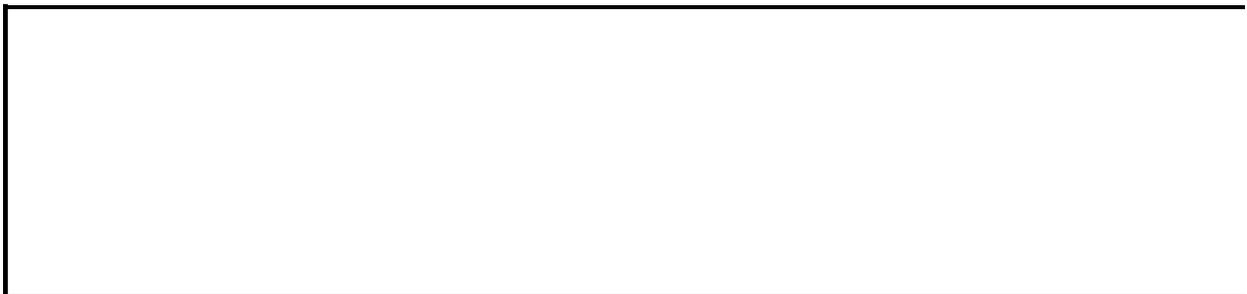
The **RightFooterPicture** property itself is read-only, but not all of its properties are read-only.

## Example

The following example adds a picture titled Sample.jpg from the C: drive to the right section of the footer. This example assumes that a file called Sample.jpg exists on the C: drive.

```
Sub InsertPicture()  
  
    With ActiveSheet.PageSetup.RightFooterPicture  
        .FileName = "C:\Sample.jpg"  
        .Height = 275.25  
        .Width = 463.5  
        .Brightness = 0.36  
        .ColorType = msoPictureGrayscale  
        .Contrast = 0.39  
        .CropBottom = -14.4  
        .CropLeft = -28.8  
        .CropRight = -14.4  
        .CropTop = 21.6  
    End With  
  
    ' Enable the image to show up in the right footer.  
    ActiveSheet.PageSetup.RightFooter = "&G"  
  
End Sub
```

**Note** It is required that "&G" be part of the **RightFooter** property string in order for the image to show up in the right footer.



# RightHeader Property

Returns or sets the right part of the header. Read/write **String**.

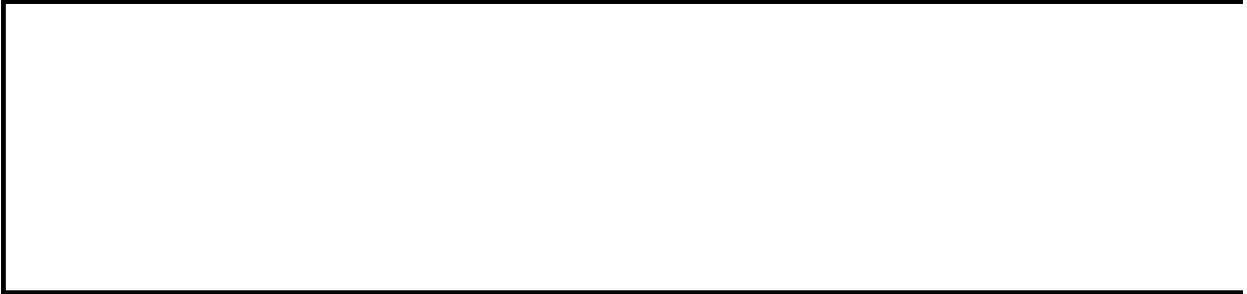
## Remarks

Special [format codes](#) can be used in the header text.

## Example

This example prints the filename in the upper-right corner of every page.

```
Worksheets("Sheet1").PageSetup.RightHeader = "&F"
```



# RightHeaderPicture Property

Returns a **Graphic** object that represents the picture for the right section of the header. Used to set attributes about the picture.

*expression*.**RightHeaderPicture**

*expression* Required. An expression that returns a [PageSetup](#) object.

## Remarks

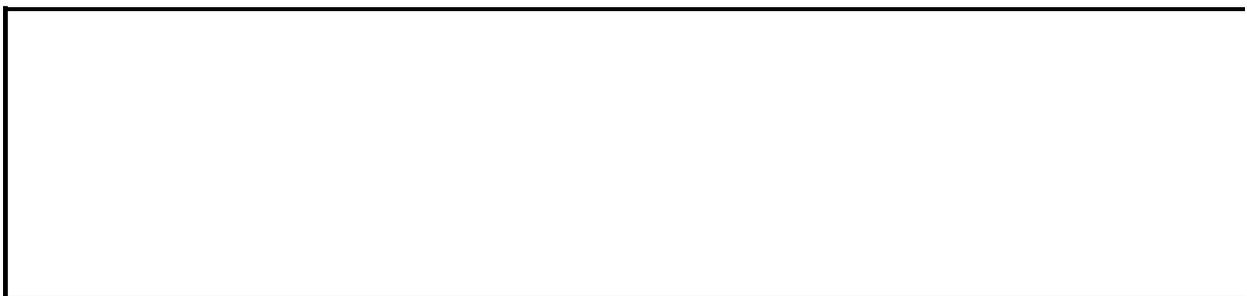
The **RightHeaderPicture** property is read-only, but the properties on it are not all read-only.

## Example

The following example adds a picture titled: Sample.jpg from the C: drive to the right section of the header. This example assumes that a file called Sample.jpg exists on the C: drive.

```
Sub InsertPicture()  
  
    With ActiveSheet.PageSetup.RightHeaderPicture  
        .FileName = "C:\Sample.jpg"  
        .Height = 275.25  
        .Width = 463.5  
        .Brightness = 0.36  
        .ColorType = msoPictureGrayscale  
        .Contrast = 0.39  
        .CropBottom = -14.4  
        .CropLeft = -28.8  
        .CropRight = -14.4  
        .CropTop = 21.6  
    End With  
  
    ' Enable the image to show up in the right header.  
    ActiveSheet.PageSetup.RightHeader = "&G"  
  
End Sub
```

**Note** It is required that "&G" be a part of the **RightHeader** property string in order for the image to show up in the right header.



[Show All](#)

# RightMargin Property

Returns or sets the size of the right margin, in [points](#). Read/write **Double**.

## Remarks

Margins are set or returned in points. Use the **InchesToPoints** method or the **CentimetersToPoints** method to convert measurements from inches or centimeters.

## Example

This example sets the right margin of Sheet1 to 1.5 inches.

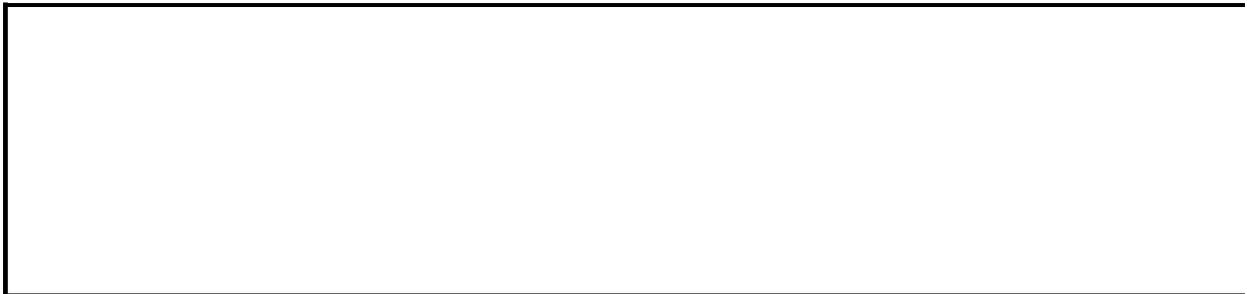
```
Worksheets("Sheet1").PageSetup.RightMargin = _  
    Application.InchesToPoints(1.5)
```

This example sets the right margin of Sheet1 to 2 centimeters.

```
Worksheets("Sheet1").PageSetup.RightMargin = _  
    Application.CentimetersToPoints(2)
```

This example displays the current right-margin setting for Sheet1.

```
marginInches = Worksheets("Sheet1").PageSetup.RightMargin / _  
    Application.InchesToPoints(1)  
MsgBox "The current right margin is " & marginInches & " inches"
```



[Show All](#)

# RobustConnect Property

Returns or sets how the PivotTable cache connects to its data source. Read/write [XIRobustConnect](#).

XIRobustConnect can be one of these XIRobustConnect constants.

**xlAlways** The cache always uses external source information (as defined by the [SourceConnectionFile](#) or [SourceDataFile](#) property) to reconnect.

**xlAsRequired** The cache uses external source info to reconnect using the [Connection](#) property.

**xlNever** The cache never uses source info to reconnect.

*expression*.**RobustConnect**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

The following example determines the setting for the cache connection and notifies the user. The example assumes a PivotTable exists on the active worksheet.

```
Sub CheckRobustConnect()  
  
    Dim pvtCache As PivotCache  
  
    Set pvtCache = Application.ActiveWorkbook.PivotCaches.Item(1)  
  
    ' Determine the connection robustness and notify user.  
    Select Case pvtCache.RobustConnect  
        Case xlAlways  
            MsgBox "The PivotTable cache is always connected to its  
        Case xlAsRequired  
            MsgBox "The PivotTable cache is connected to its source  
        Case xlNever  
            MsgBox "The PivotTable cache is never connected to its s  
    End Select  
  
End Sub
```



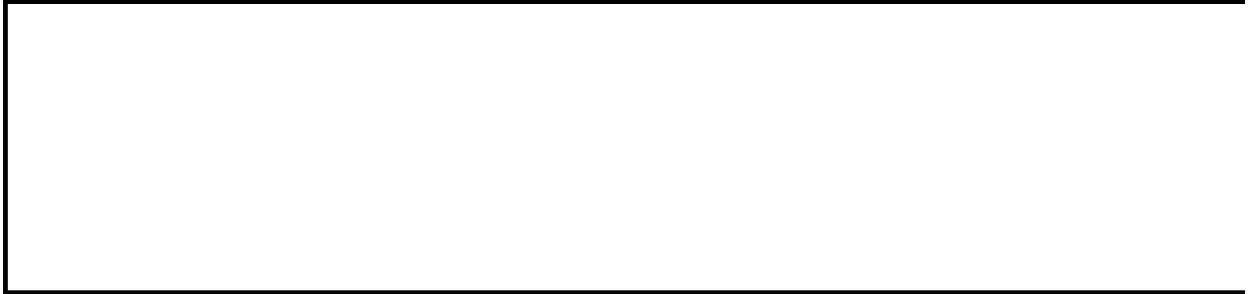
# RollZoom Property

**True** if the IntelliMouse zooms instead of scrolling. Read/write **Boolean**.

## Example

This example enables the IntelliMouse to zoom instead of scroll.

```
Application.RollZoom = True
```



# Root Property

Returns the root **DiagramNode** object which the root diagram node belongs.

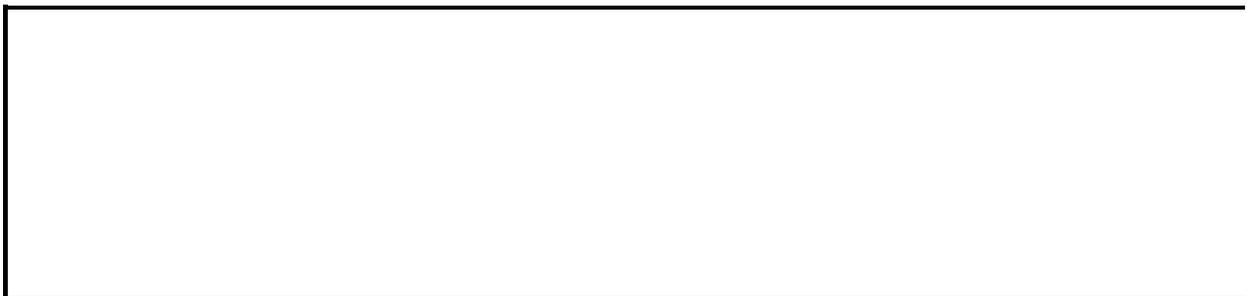
*expression*.**Root**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

The following example creates an organization chart and adds child nodes to the root diagram node.

```
Sub AddChildNodesToRoot()  
  
    Dim nodDiagram As DiagramNode  
    Dim shDiagram As Shape  
    Dim intCount As Integer  
  
    Set shDiagram = ActiveSheet.Shapes.AddDiagram _  
        (Type:=msoDiagramOrgChart, Left:=10, Top:=15, _  
        Width:=400, Height:=475)  
  
    ' Add the first node to the diagram.  
    shDiagram.DiagramNode.Children.AddNode  
  
    ' Establish the first node as the root.  
    Set nodDiagram = shDiagram.DiagramNode.Root  
  
    ' Add three nodes to the diagram.  
    For intCount = 1 To 3  
        nodDiagram.Children.AddNode  
    Next intCount  
  
End Sub
```



[Show All](#)

# RootElementName Property

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Returns a **String** that represents the name of the [root element](#) for the specified XML schema map. Read-only.

*expression*.**RootElementName**

*expression* Required. An expression that returns an [XmlMap](#) object.



[Show All](#)

# RootElementNamespace Property

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Returns an [XmlNamespace](#) object that represents the [root element](#) for the specified XML schema map. Read-only.

*expression*.**RootElementNamespace**

*expression* Required. An expression that returns an [XmlMap](#) object.



[Show All](#)

# RotatedChars Property

**True** if characters in the specified WordArt are rotated 90 degrees relative to the WordArt's bounding shape. **False** if characters in the specified WordArt retain their original orientation relative to the bounding shape. Read/write [MsoTriState](#).

MsoTriState can be one of these MsoTriState constants.

**msoCTrue**

**msoFalse** Characters in the specified WordArt retain their original orientation relative to the bounding shape.

**msoTriStateMixed**

**msoTriStateToggle**

**msoTrue** Characters in the specified WordArt are rotated 90 degrees relative to the WordArt's bounding shape.

*expression*.**RotatedChars**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

If the WordArt has horizontal text, setting the **RotatedChars** property to **msoTrue** rotates the characters 90 degrees counterclockwise. If the WordArt has vertical text, setting the **RotatedChars** property to **msoFalse** rotates the characters 90 degrees clockwise. Use the **ToggleVerticalText** method to switch between horizontal and vertical text flow.

The [Flip](#) method and [Rotation](#) property of the [Shape](#) object and the **RotatedChars** property and [ToggleVerticalText](#) method of the **TextEffectFormat** object all affect the character orientation and direction of text flow in a **Shape** object that represents WordArt. You may have to experiment to find out how to combine the effects of these properties and methods to get the result you want.

## Example

This example adds WordArt that contains the text "Test" to myDocument and rotates the characters 90 degrees counterclockwise.

```
Set myDocument = Worksheets(1)
Set newWordArt = myDocument.Shapes.AddTextEffect( _
    PresetTextEffect:=msoTextEffect1, Text:="Test", _
    FontName:="Arial Black", FontSize:=36, _
    FontBold:=False, FontItalic:=False, Left:=10, _
    Top:=10)
newWordArt.TextEffect.RotatedChars = msoTrue
```



# Rotation Property

**Chart** object: Returns or sets the rotation of the 3-D chart view (the rotation of the plot area around the z-axis, in degrees). The value of this property must be from 0 to 360, except for 3-D bar charts, where the value must be from 0 to 44. The default value is 20. Applies only to 3-D charts. Read/write **Variant**.

**Shape** or **ShapeRange** object: Returns or sets the rotation of the shape, in degrees. Read/write **Single**.

## Example

This example sets the rotation of Chart1 to 30 degrees. The example should be run on a 3-D chart.

```
Charts("Chart1").Rotation = 30
```



# RotationX Property

Returns or sets the rotation of the extruded shape around the x-axis in degrees. Can be a value from – 90 through 90. A positive value indicates upward rotation; a negative value indicates downward rotation. Read/write **Single**.

*expression*.**RotationX**

*expression* Required. An expression that returns one of the objects in the Applies To list.

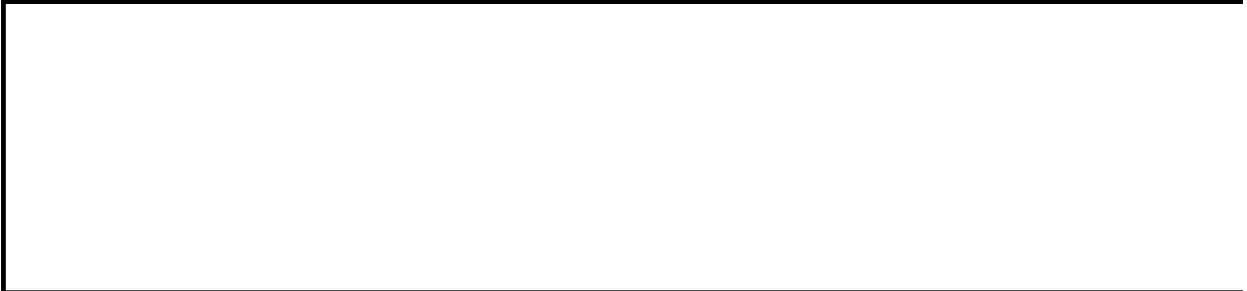
## Remarks

To set the rotation of the extruded shape around the y-axis, use the [RotationY](#) property of the ThreeDFormat object. To set the rotation of the extruded shape around the z-axis, use the [Rotation](#) property of the [Shape](#) object. To change the direction of the extrusion's sweep path without rotating the front face of the extrusion, use the [SetExtrusionDirection](#) method.

## Example

This example adds three identical extruded ovals to myDocument and sets their rotation around the x-axis to – 30, 0, and 30 degrees, respectively.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes
    With .AddShape(msoShapeOval, 30, 30, 50, 25).ThreeD
        .Visible = True
        .RotationX = -30
    End With
    With .AddShape(msoShapeOval, 30, 70, 50, 25).ThreeD
        .Visible = True
        .RotationX = 0
    End With
    With .AddShape(msoShapeOval, 30, 110, 50, 25).ThreeD
        .Visible = True
        .RotationX = 30
    End With
End With
```



# RotationY Property

Returns or sets the rotation of the extruded shape around the y-axis in degrees. Can be a value from – 90 through 90. A positive value indicates rotation to the left; a negative value indicates rotation to the right. Read/write **Single**.

*expression*.**RotationY**

*expression* Required. An expression that returns one of the objects in the Applies To list.

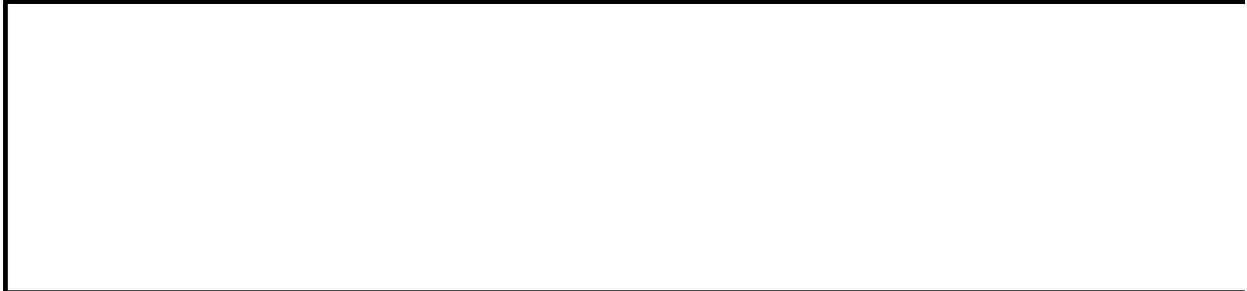
## Remarks

To set the rotation of the extruded shape around the x-axis, use the [RotationX](#) property of the **ThreeDFormat** object. To set the rotation of the extruded shape around the z-axis, use the [Rotation](#) property of the [Shape](#) object. To change the direction of the extrusion's sweep path without rotating the front face of the extrusion, use the [SetExtrusionDirection](#) method.

## Example

This example adds three identical extruded ovals to myDocument and sets their rotation around the y-axis to – 30, 0, and 30 degrees, respectively.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes
    With .AddShape(msoShapeOval, 30, 30, 50, 25).ThreeD
        .Visible = True
        .RotationY = -30
    End With
    With .AddShape(msoShapeOval, 30, 70, 50, 25).ThreeD
        .Visible = True
        .RotationY = 0
    End With
    With .AddShape(msoShapeOval, 30, 110, 50, 25).ThreeD
        .Visible = True
        .RotationY = 30
    End With
End With
```



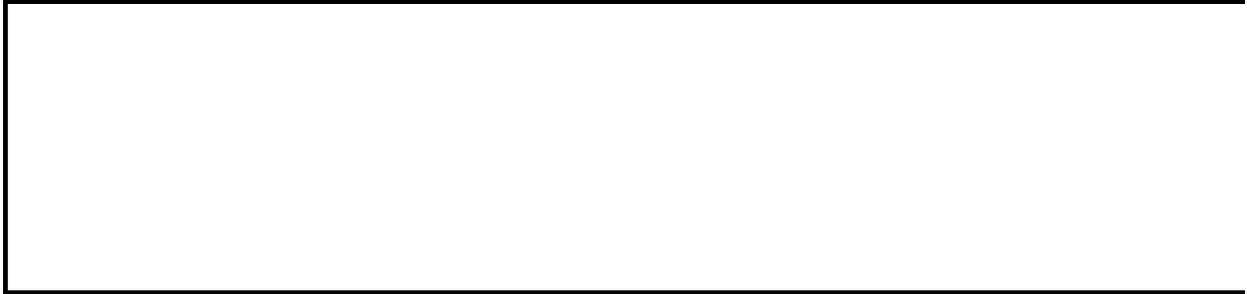
# RoundedCorners Property

**True** if the embedded chart has rounded corners. Read/write **Boolean**.

## Example

This example adds rounded corners to embedded chart one on Sheet1.

```
worksheets("Sheet1").ChartObjects(1).RoundedCorners = True
```



# Routed Property

**True** if the workbook has been routed to the next recipient. **False** if the workbook needs to be routed. Read-only **Boolean**.

## Remarks

If the workbook wasn't routed to the current recipient, this property is always **False** (for example, if the document has no routing slip, or if a routing slip was just created).

## Example

This example sends the workbook to the next recipient.

```
If ActiveWorkbook.HasRoutingSlip And _  
    Not ActiveWorkbook.Routed Then  
    ActiveWorkbook.Route  
End If
```



# RoutingSlip Property

Returns a [RoutingSlip](#) object that represents the routing slip for the workbook. Reading this property if there's no routing slip causes an error (check the **HasRoutingSlip** property first). Read-only.

## Example

This example creates a routing slip for Book1.xls and then sends the workbook to three recipients, one after another.

```
Workbooks("BOOK1.XLS").HasRoutingSlip = True
With Workbooks("BOOK1.XLS").RoutingSlip
    .Delivery = xlOneAfterAnother
    .Recipients = Array("Adam Bendel", _
        "Jean Selva", "Bernard Gabor")
    .Subject = "Here is BOOK1.XLS"
    .Message = "Here is the workbook. What do you think?"
End With
Workbooks("BOOK1.XLS").Route
```



# Row Property

Returns the number of the first row of the first area in the range. Read-only  
**Long.**

## Example

This example sets the row height of every other row on Sheet1 to 4 points.

```
For Each rw In Worksheets("Sheet1").Rows
    If rw.Row Mod 2 = 0 Then
        rw.RowHeight = 4
    End If
Next rw
```



# RowColSettings Property

**True** if the custom view includes settings for hidden rows and columns (including filter information). Read-only **Boolean**.

## Example

This example creates a list of the custom views in the active workbook and their print settings and row and column settings.

```
With Worksheets(1)
    .Cells(1,1).Value = "Name"
    .Cells(1,2).Value = "Print Settings"
    .Cells(1,3).Value = "RowColSettings"
    rw = 0
    For Each v In ActiveWorkbook.CustomViews
        rw = rw + 1
        .Cells(rw, 1).Value = v.Name
        .Cells(rw, 2).Value = v.PrintSettings
        .Cells(rw, 3).Value = v.RowColSettings
    Next
End With
```



# RowFields Property

Returns an object that represents either a single field in a PivotTable report (a [PivotField](#) object) or a collection of all the fields (a [PivotFields](#) object) that are currently showing as row fields. Read-only.

*expression*.**RowFields**(*Index*)

*expression* Required. An expression that returns a **PivotTable** object.

*Index* Optional **Variant**. The name or number of the field to be returned (can be an array to specify more than one field).

## Example

This example adds the PivotTable report's row field names to a list on a new worksheet.

```
Set nwSheet = Worksheets.Add  
nwSheet.Activate  
Set pvtTable = Worksheets("Sheet2").Range("A1").PivotTable  
rw = 0  
For Each pvtField In pvtTable.RowFields  
    rw = rw + 1  
    nwSheet.Cells(rw, 1).Value = pvtField.Name  
Next pvtField
```



# RowGrand Property

**True** if the PivotTable report shows grand totals for rows. Read/write **Boolean**.

## Example

This example sets the PivotTable report to show grand totals for rows.

```
Set pvtTable = Worksheets("Sheet1").Range("A3").PivotTable  
pvtTable.RowGrand = True
```



[Show All](#)

# RowHeight Property

Returns the height of all the rows in the range specified, measured in [points](#).

Returns **Null** if the rows in the specified range aren't all the same height.

Read/write **Variant**.

## Remarks

For a single row, the value of the **Height** property is equal to the value of the **RowHeight** property. However, you can also use the **Height** property to return the total height of a range of cells.

Other differences between **RowHeight** and **Height** include the following:

- **Height** is read-only.
- If you return the **RowHeight** property of several rows, you will either get the row height of each of the rows (if all the rows are the same height) or **Null** (if they're different heights). If you return the **Height** property of several rows, you will get the total height of all the rows.

## Example

This example doubles the height of row one on Sheet1.

```
With Worksheets("Sheet1").Rows(1)  
    .RowHeight = .RowHeight * 2  
End With
```



# RowItems Property

Returns a [PivotItemList](#) collection that corresponds to the items on the category axis that represent the selected cell.

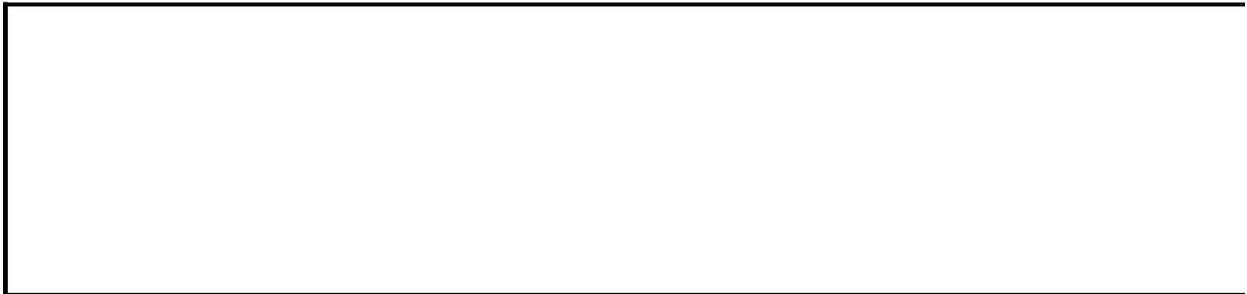
*expression*.**RowItems**

*expression* Required. An expression that returns a [PivotCell](#) object.

## Example

This example determines if the data item in cell B5 is under the Inventory item in the first row field and notifies the user. The example assumes a PivotTable exists on the active worksheet and that column B of the worksheet contains a row item of the PivotTable.

```
Sub CheckRowItems()  
    ' Determine if there is a match between the item and row field.  
    If Application.Range("B5").PivotCell.RowItems.Item(1) = "Inventory"  
        MsgBox "Cell B5 is a member of the 'Inventory' row field."  
    Else  
        MsgBox "Cell B5 is not a member of the 'Inventory' row field."  
    End If  
End Sub
```



# RowNumbers Property

**True** if row numbers are added as the first column of the specified query table.  
Read/write **Boolean**.

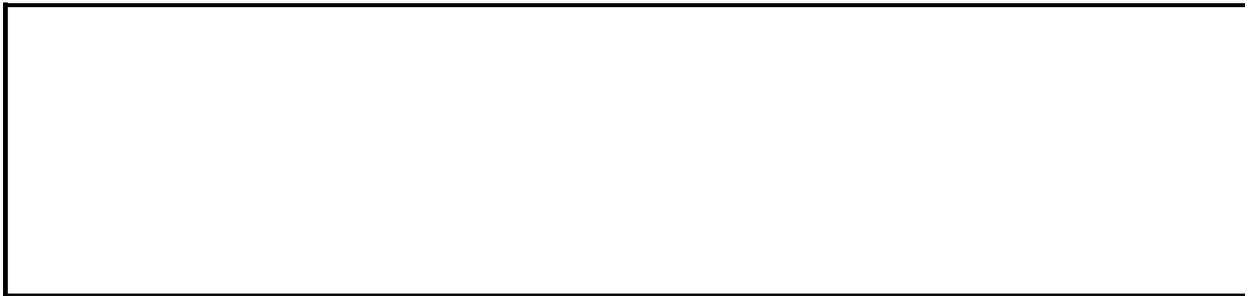
## Remarks

Setting this property to **True** doesn't immediately cause row numbers to appear. The row numbers appear the next time the query table is refreshed, and they're reconfigured every time the query table is refreshed.

## Example

This example adds row numbers and field names to the query table.

```
With Worksheets(1).QueryTables("ExternalData1")  
    .RowNumbers = True  
    .FieldNames = True  
    .Refresh  
End With
```



# RowRange Property

Returns a [Range](#) object that represents the range including the row area on the PivotTable report. Read-only.

## Example

This example selects the row headers on the PivotTable report.

```
Worksheets("Sheet1").Activate  
Range("A3").Select  
ActiveCell.PivotTable.RowRange.Select
```



# Rows Property

For an **Application** object, returns a [Range](#) object that represents all the rows on the active worksheet. If the active document isn't a worksheet, the **Rows** property fails. For a **Range** object, returns a **Range** object that represents the rows in the specified range. For a **Worksheet** object, returns a **Range** object that represents all the rows on the specified worksheet. Read-only **Range** object.

## Remarks

For information about returning a single member of a collection, see [Returning an Object from a Collection](#).

Using this property without an object qualifier is equivalent to using `ActiveSheet.Rows`.

When applied to a **Range** object that's a multiple selection, this property returns rows from only the first area of the range. For example, if the **Range** object has two areas— A1:B2 and C3:D4— `Selection.Rows.Count` returns 2, not 4. To use this property on a range that may contain a multiple selection, test `Areas.Count` to determine whether the range is a multiple selection. If it is, loop over each area in the range, as shown in the third example.

## Example

This example deletes row three on Sheet1.

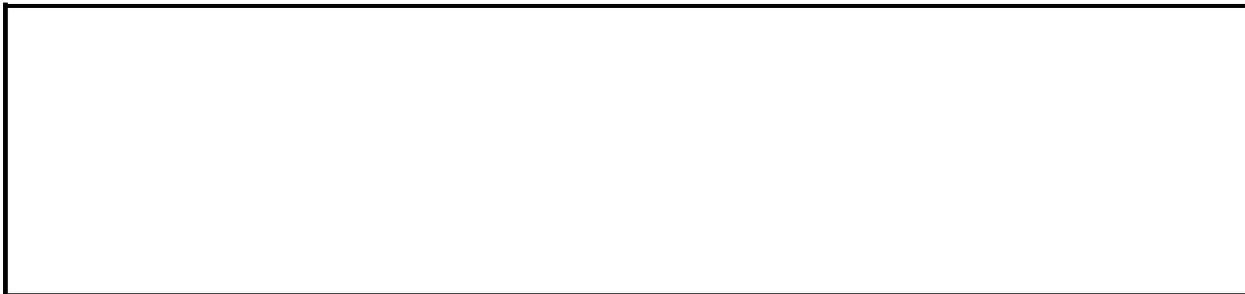
```
Worksheets("Sheet1").Rows(3).Delete
```

This example deletes rows in the current region on worksheet one where the value of cell one in the row is the same as the value in cell one in the previous row.

```
For Each rw In Worksheets(1).Cells(1, 1).CurrentRegion.Rows
    this = rw.Cells(1, 1).Value
    If this = last Then rw.Delete
    last = this
Next
```

This example displays the number of rows in the selection on Sheet1. If more than one area is selected, the example loops through each area.

```
Worksheets("Sheet1").Activate
areaCount = Selection.Areas.Count
If areaCount <= 1 Then
    MsgBox "The selection contains " & _
        Selection.Rows.Count & " rows."
Else
    i = 1
    For Each a In Selection.Areas
        MsgBox "Area " & i & " of the selection contains " & _
            a.Rows.Count & " rows."
        i = i + 1
    Next a
End If
```



# RTD Property

Returns an [RTD](#) object.

*expression*.**RTD**

*expression* Required. An expression that returns an [Application](#) object.



# Saved Property

**True** if no changes have been made to the specified workbook since it was last saved. Read/write **Boolean**.

## Remarks

If a workbook has never been saved, its [Path](#) property returns an empty string ("").

You can set this property to **True** if you want to close a modified workbook without either saving it or being prompted to save it.

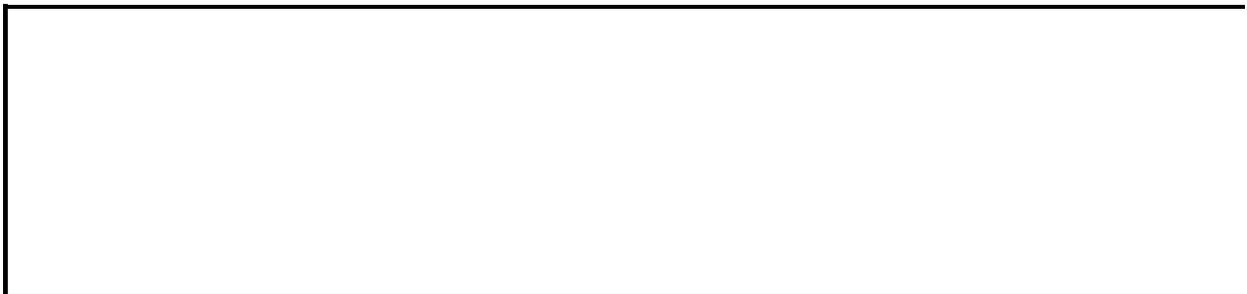
## Example

This example displays a message if the active workbook contains unsaved changes.

```
If Not ActiveWorkbook.Saved Then  
    MsgBox "This workbook contains unsaved changes."  
End If
```

This example closes the workbook that contains the example code and discards any changes to the workbook by setting the **Saved** property to **True**.

```
ThisWorkbook.Saved = True  
ThisWorkbook.Close
```



[Show All](#)

# SaveData Property

**True** if data for the PivotTable report is saved with the workbook. **False** if only the report definition is saved. Read/write **Boolean**.

## Remarks

For [OLAP](#) data sources, this property is always set to **False**.

## Example

This example sets the PivotTable report to save data with the workbook.

```
Set pvtTable = Worksheets("Sheet1").Range("A3").PivotTable  
pvtTable.SaveData = True
```



# SaveDataSourceDefinition Property

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

**True** if the data source definition of the specified XML schema map is saved with the workbook. The default value is **True**. Read/write **Boolean**.

*expression*.**SaveDataSourceDefinition**

*expression* Required. An expression that returns an [XmlMap](#) object.



# SaveHiddenData Property

**True** if data outside of the specified range is saved when you save the document as a Web page. This data may be necessary for maintaining formulas. **False** if data outside of the specified range is not saved with the Web page. The default value is **True**. Read/write **Boolean**.

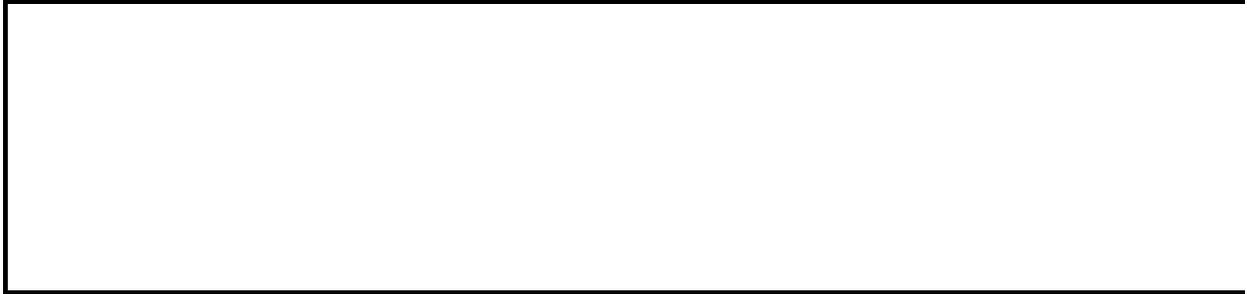
## Remarks

If you choose not to save data outside of the specified range, references to that data in the formula are converted to static values. If the data is in another sheet or workbook, the result of the formula is saved as a static value.

## Example

This example prevents hidden data from being saved with Web pages.

```
Application.DefaultWebOptions.SaveHiddenData = False
```



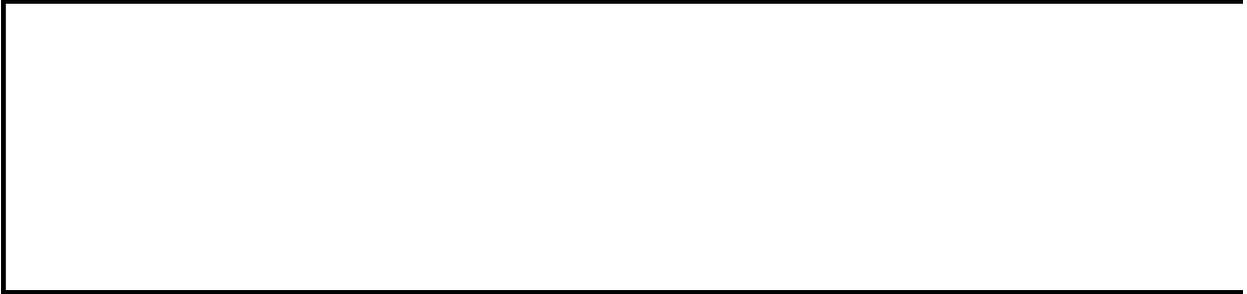
# SaveLinkValues Property

**True** if Microsoft Excel saves external link values with the workbook.  
Read/write **Boolean**.

## Example

This example causes Microsoft Excel to save external link values with the active workbook.

```
ActiveWorkbook.SaveLinkValues = True
```



# SaveNewWebPagesAsWebArchives Property

**True** if new Web pages can be saved as Web archives. Read/write **Boolean**.

*expression*.**SaveNewWebPagesAsWebArchives**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel determines the settings for saving new Web pages as Web archives and notifies the user.

```
Sub DetermineSettings()  
    ' Determine settings and notify user.  
    If Application.DefaultWebOptions.SaveNewWebPagesAsWebArchives =  
        MsgBox "New Web pages will be saved as Web archives."  
    Else  
        MsgBox "New Web pages will not be saved as Web archives."  
    End If  
End Sub
```



# SavePassword Property

**True** if password information in an ODBC connection string is saved with the specified query. **False** if the password is removed. Read/write **Boolean**.

## **Remarks**

This property affects only ODBC queries.

## Example

This example causes password information to be removed from the ODBC connection string whenever query table one is saved.

```
Worksheets(1).QueryTables(1).SavePassword = False
```



[Show All](#)

# ScaleType Property

Returns or sets the value axis scale type. Read/write [XlScaleType](#).

XlScaleType can be one of these XlScaleType constants.

**xlScaleLinear**

**xlScaleLogarithmic**

*expression*.**ScaleType**

*expression* Required. An expression that returns one of the objects in the Applies To list.

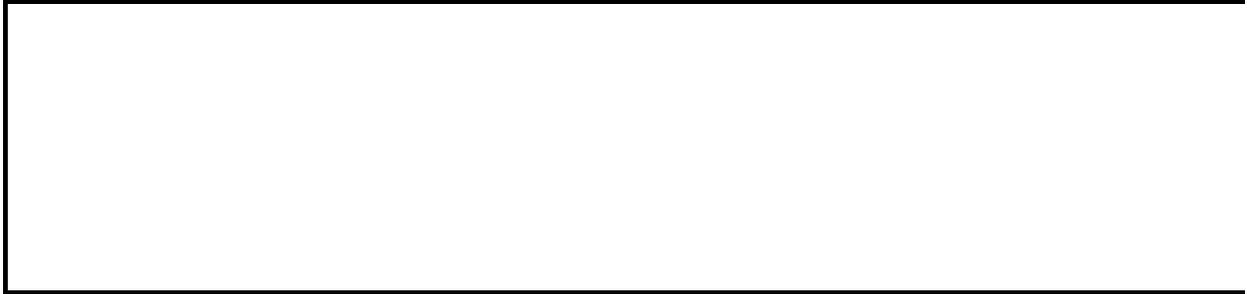
## **Remarks**

A logarithmic scale uses base 10 logarithms.

## Example

This example sets the value axis in Chart1 to use a logarithmic scale.

```
Charts("Chart1").Axes(xlValue).ScaleType = xlScaleLogarithmic
```



[Show All](#)

# Schemas Property

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Returns an [XmlSchemas](#) collection that represents the schemas that the specified [XmlMap](#) object contains. Read-only.

*expression*.**Schemas**

*expression* Required. An expression that returns an **XmlMap** object.

## Remarks

The first item in the returned **XmlSchemas** collection is the schema that is used as the [root element](#) of the specified XML schema map.

---

---

---

[Show All](#)

# SchemeColor Property

 [SchemeColor property as it applies to the \*\*ColorFormat\*\* object.](#)

Returns or sets the color of a **Color** object as an index in the current color scheme. Read/write **Integer**.

*expression*.**SchemeColor**

*expression* Required. An expression that returns one of the above objects.

 [SchemeColor property as it applies to the \*\*ChartColorFormat\*\* object.](#)

Returns or sets the color of a **Color** object as an index in the current color scheme. Read/write **Long**.

*expression*.**SchemeColor**

*expression* Required. An expression that returns one of the above objects.

## Example

This example sets the foreground color, background color, and gradient for the chart area fill on chart one.

```
With Charts(1).ChartArea.Fill
    .Visible = True
    .ForeColor.SchemeColor = 15
    .BackColor.SchemeColor = 17
    .TwoColorGradient msoGradientHorizontal, 1
End With
```



[Show All](#)

# ScreenSize Property

Returns or sets the ideal minimum screen size (width by height, in pixels) that you should use when viewing the saved document in a Web browser. Can be one of the **MsoScreenSize** constants listed below. The default constant is **msoScreenSize800x600**. Read/write [MsoScreenSize](#).

MsoScreenSize can be one of these MsoScreenSize constants.

**msoScreenSize1152x882**

**msoScreenSize1280x1024**

**msoScreenSize1800x1440**

**msoScreenSize544x376**

**msoScreenSize720x512**

**msoScreenSize1024x768**

**msoScreenSize1152x900**

**msoScreenSize1600x1200**

**msoScreenSize1920x1200**

**msoScreenSize640x480**

**msoScreenSize800x600**

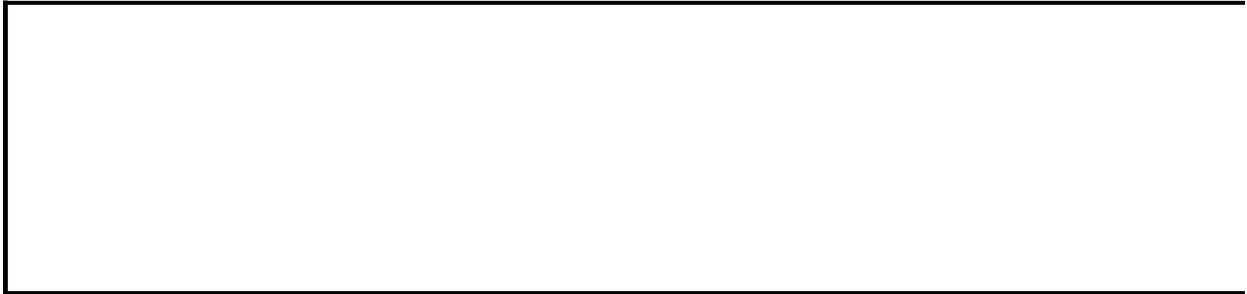
*expression*.**ScreenSize**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example sets the target screen size at 800x600 pixels.

```
Application.DefaultWebOptions.ScreenSize = _  
    msoScreenSize800x600
```



# ScreenTip Property

Returns or sets the ScreenTip text for the specified hyperlink. Read/write **String**.

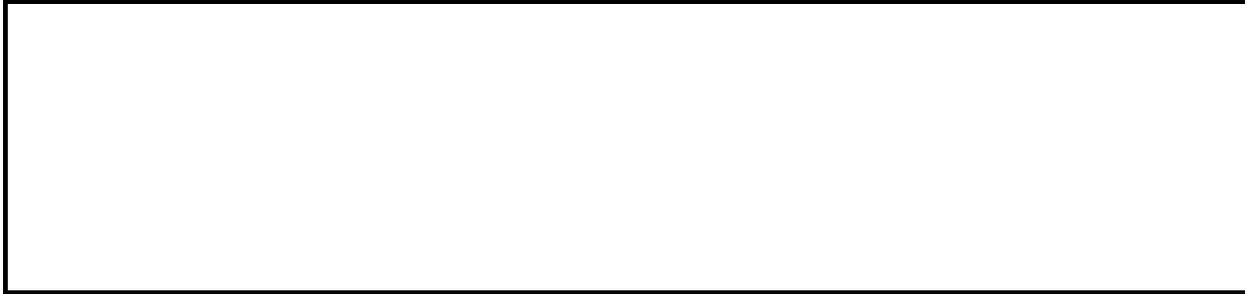
## **Remarks**

After the document has been saved to a Web page, the ScreenTip text may appear (for example) when the mouse pointer is positioned over the hyperlink while the document is being viewed in a Web browser. Some Web browsers may not support ScreenTips.

## Example

This example sets the screen tip for the first hyperlink on the active worksheet.

```
ActiveSheet.Hyperlinks(1).ScreenTip = "Return to the home page"
```



# ScreenUpdating Property

**True** if screen updating is turned on. Read/write **Boolean**.

## Remarks

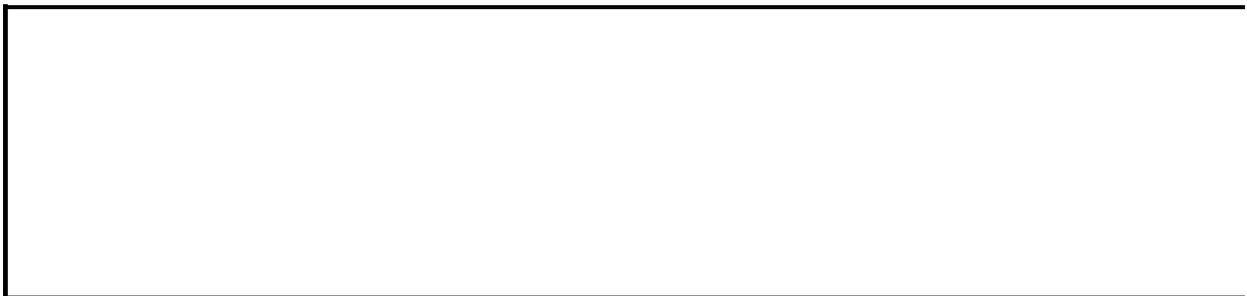
Turn screen updating off to speed up your macro code. You won't be able to see what the macro is doing, but it will run faster.

Remember to set the **ScreenUpdating** property back to **True** when your macro ends.

## Example

This example demonstrates how turning off screen updating can make your code run faster. The example hides every other column on Sheet1, while keeping track of the time it takes to do so. The first time the example hides the columns, screen updating is turned on; the second time, screen updating is turned off. When you run this example, you can compare the respective running times, which are displayed in the message box.

```
Dim elapsedTime(2)
Application.ScreenUpdating = True
For i = 1 To 2
    If i = 2 Then Application.ScreenUpdating = False
    startTime = Time
    Worksheets("Sheet1").Activate
    For Each c In ActiveSheet.Columns
        If c.Column Mod 2 = 0 Then
            c.Hidden = True
        End If
    Next c
    stopTime = Time
    elapsedTime(i) = (stopTime - startTime) * 24 * 60 * 60
Next i
Application.ScreenUpdating = True
MsgBox "Elapsed time, screen updating on: " & elapsedTime(1) & _
    " sec." & Chr(13) & _
    "Elapsed time, screen updating off: " & elapsedTime(2) & _
    " sec."
```



# Script Property

Returns the **Script** object, which represents a block of script or code on the specified Web page. If the page contains no script, nothing is returned.

*expression*.**Script**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example displays the type of scripting language used in the first shape on the active worksheet.

```
Set objScr = ActiveSheet.Shapes(1).Script
If Not (objScr Is Nothing) Then
    Select Case objScr.Language
        Case msoScriptLanguageVisualBasic
            MsgBox "VBScript"
        Case msoScriptLanguageJava
            MsgBox "JavaScript"
        Case msoScriptLanguageASP
            MsgBox "Active Server Pages"
        Case Else
            MsgBox "Other scripting language"
    End Select
End If
```



# Scripts Property

Returns the **Scripts** collection, which contains **Script** objects representing blocks of script or code in the specified document when it's saved as a Web page.

*expression*.**Scripts**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example adds a new VBScript block to the **Scripts** collection on the active worksheet.

```
Set objScrs = ActiveSheet.Scripts  
Set objNewScr = objScrs.Add  
objNewScr.Language = msoScriptLanguageVisualBasic
```



# ScrollArea Property

Returns or sets the range where scrolling is allowed, as an A1-style range reference. Cells outside the scroll area cannot be selected. Read/write **String**.

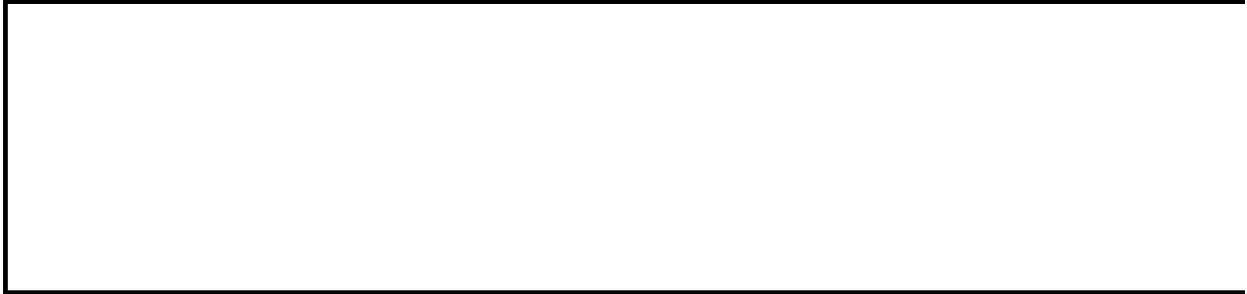
## Remarks

Set this property to the empty string ("" ) to enable cell selection for the entire sheet.

## Example

This example sets the scroll area for worksheet one.

```
Worksheets(1).ScrollArea = "a1:f10"
```



# ScrollColumn Property

Returns or sets the number of the leftmost column in the pane or window.  
Read/write **Long**.

## Remarks

If the window is split, the **ScrollColumn** property of the **Window** object refers to the upper-left pane. If the panes are frozen, the **ScrollColumn** property of the **Window** object excludes the frozen areas.

## Example

This example moves column three so that it's the leftmost column in the window.

```
Worksheets("Sheet1").Activate  
ActiveWindow.ScrollColumn = 3
```



# ScrollRow Property

Returns or sets the number of the row that appears at the top of the pane or window. Read/write **Long**.

## Remarks

If the window is split, the **ScrollRow** property of the **Window** object refers to the upper-left pane. If the panes are frozen, the **ScrollRow** property of the **Window** object excludes the frozen areas.

## Example

This example moves row ten to the top of the window.

```
Worksheets("Sheet1").Activate  
ActiveWindow.ScrollRow = 10
```



# SecondaryPlot Property

**True** if the point is in the secondary section of either a pie of pie chart or a bar of pie chart. Applies only to points on pie of pie charts or bar of pie charts.

Read/write **Boolean**.

## Example

This example must be run on either a pie of pie chart or a bar of pie chart. The example moves point four to the secondary section of the chart.

```
With Worksheets(1).ChartObjects(1).Chart.SeriesCollection(1)  
    .Points(4).SecondaryPlot = True  
End With
```



# SecondPlotSize Property

Returns or sets the size of the secondary section of either a pie of pie chart or a bar of pie chart, as a percentage of the size of the primary pie. Can be a value from 5 to 200. Read/write **Long**.

## Example

This example must be run on either a pie of pie chart or a bar of pie chart. The example splits the two sections of the chart by value, combining all values under 10 in the primary pie and displaying them in the secondary section. The secondary section is 50 percent of the size of the primary pie.

```
With Worksheets(1).ChartObjects(1).Chart.ChartGroups(1)
    .SplitType = xlSplitByValue
    .SplitValue = 10
    .VaryByCategories = True
    .SecondPlotSize = 50
End With
```



[Show All](#)

# SegmentType Property

Returns a value that indicates whether the segment associated with the specified node is straight or curved. If the specified node is a control point for a curved segment, this property returns **msoSegmentCurve**. Read-only [MsoSegmentType](#).

MsoSegmentType can be one of these MsoSegmentType constants.

**msoSegmentCurve**

**msoSegmentLine**

*expression*.**SegmentType**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Use the [SetSegmentType](#) method to set the value of this property.

## Example

This example changes all straight segments to curved segments in shape three on myDocument. Shape three must be a freeform drawing.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(3).Nodes
    n = 1
    While n <= .Count
        If .Item(n).SegmentType = msoSegmentLine Then
            .SetSegmentType n, msoSegmentCurve
        End If
        n = n + 1
    Wend
End With
```



# SelectedSheets Property

Returns a [Sheets](#) collection that represents all the selected sheets in the specified window. Read-only.

For information about returning a single member of a collection, see [Returning an Object from a Collection](#).

## Example

This example displays a message if Sheet1 is selected in Book1.xls.

```
For Each sh In Workbooks("BOOK1.XLS").Windows(1).SelectedSheets
    If sh.Name = "Sheet1" Then
        MsgBox "Sheet1 is selected"
    Exit For
End If
Next
```



# Selection Property

Returns the selected object in the active window, for an **Application** object, and a specified window, for a **Windows** object.

## Remarks

The returned object type depends on the current selection (for example, if a cell is selected, this property returns a [Range](#) object). The **Selection** property returns **Nothing** if nothing is selected.

Using this property with no object qualifier is equivalent to using `Application.Selection`.

## Example

This example clears the selection on Sheet1 (assuming that the selection is a range of cells).

```
Worksheets("Sheet1").Activate  
Selection.Clear
```

This example displays the Visual Basic object type of the selection.

```
Worksheets("Sheet1").Activate  
MsgBox "The selection object type is " & TypeName(Selection)
```



[Show All](#)

# SelectionMode Property

Returns or sets the PivotTable report structured selection mode. Read/write [XIPTSelectionMode](#).

XIPTSelectionMode can be one of these XIPTSelectionMode constants.

**xlBlanks**

**xlButton**

**xlDataAndLabel**

**xlDataOnly**

**xlFirstRow**

**xlLabelOnly**

**xlOrigin**

*expression*.**SelectionMode**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

If the PivotTable field isn't in outline form, specifying the sum of any of the constants and **xlFirstRow** is equivalent to specifying the constant alone.

## Example

This example enables structured selection mode and then sets the first PivotTable report on worksheet one to allow only data to be selected.

```
Application.PivotTableSelection = True  
Worksheets(1).PivotTables(1).SelectionMode = xlDataOnly
```

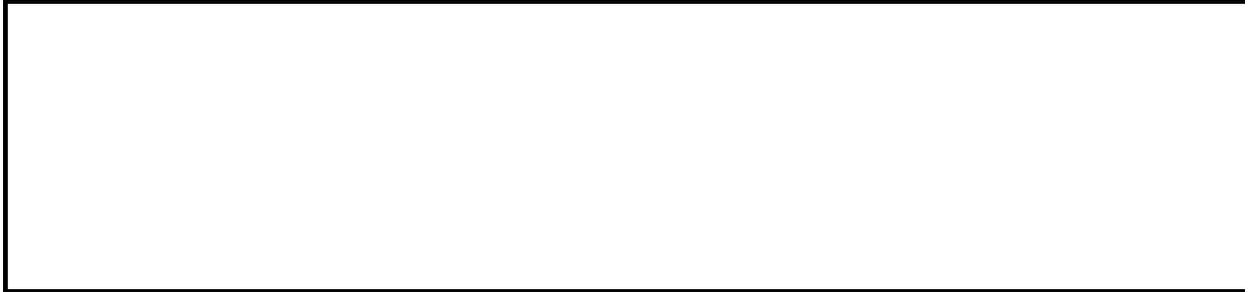
In this example, the PivotTable report is in outline form.

```
Application.PivotTableSelection = True  
Worksheets(1).PivotTables(1).SelectionMode = _  
    xlDataOnly + xlFirstRow
```



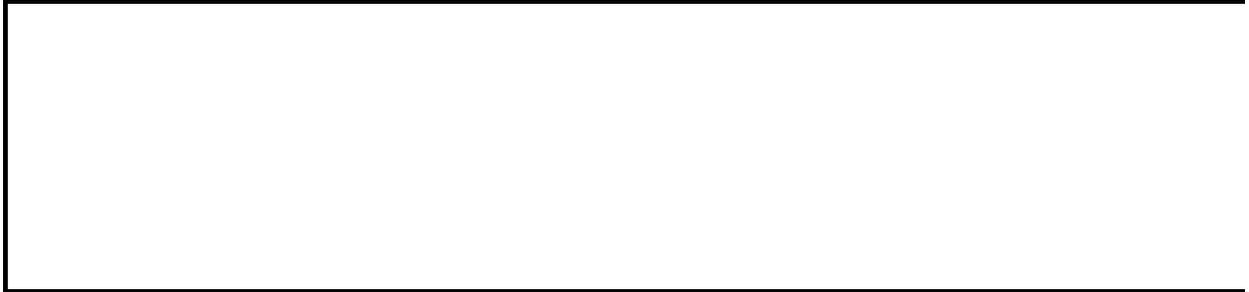
# SendDateTime Property

You have requested Help for a Visual Basic keyword used only on the Macintosh. For information about this keyword, consult the language reference Help included with Microsoft Office Macintosh Edition.



# Sender Property

You have requested Help for a Visual Basic keyword used only on the Macintosh. For information about this keyword, consult the language reference Help included with Microsoft Office Macintosh Edition.



# Separator Property

Sets or returns a **Variant** representing the separator used for the data labels on a chart. Read/write.

*expression*.**Separator**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

The chart must first be active before you can access the data labels programmatically; otherwise a run-time error occurs.

## Example

This example sets the data label separator for the first series on the first chart to a semicolon. This example assumes a chart exists on the active worksheet.

```
Sub ChangeSeparator()  
    ActiveSheet.ChartObjects(1).Activate  
    ActiveChart.SeriesCollection(1) _  
        .DataLabels.Separator = ";"  
End Sub
```



# SeriesLines Property

Returns a [SeriesLines](#) object that represents the series lines for a stacked bar chart or a stacked column chart. Applies only to stacked bar and stacked column charts. Read-only.

## Example

This example turns on series lines for chart group one in Chart1 and then sets their line style, weight, and color. The example should be run on a 2-D stacked column chart that has two or more series.

```
With Charts("Chart1").ChartGroups(1)
    .HasSeriesLines = True
    With .SeriesLines.Border
        .LineStyle = xlThin
        .Weight = xlMedium
        .ColorIndex = 3
    End With
End With
```



[Show All](#)

# ServerBased Property

**True** if the data source for the specified PivotTable report is external and only the items matching the page field selection are retrieved. Read/write **Boolean**.

This property doesn't apply to [OLAP](#) data sources and is always **False**.

When this property is **True**, only records in the database that match the selected page field item are retrieved. From then on, whenever the user changes the page field selection, the newly selected page field item is passed to the query as a parameter, and the cache is refreshed.

This property cannot be set if any of the following conditions are true:

- The field is grouped.
- The data source isn't external.
- The cache is shared by two or more PivotTable reports.
- The field is a data type that cannot be server based (a memo field or an OLE object).

## Example

This example lists all the server-based page fields.

```
For Each fld in ActiveSheet.PivotTables(1).PageFields
  If fld.ServerBased = True Then
    r = r + 1
    Worksheets(2).Cells(r, 1).Value = fld.Name
  End If
Next
```



# Shadow Property

**True** if the font is a shadow font or if the object has a shadow. Read/write **Boolean**.

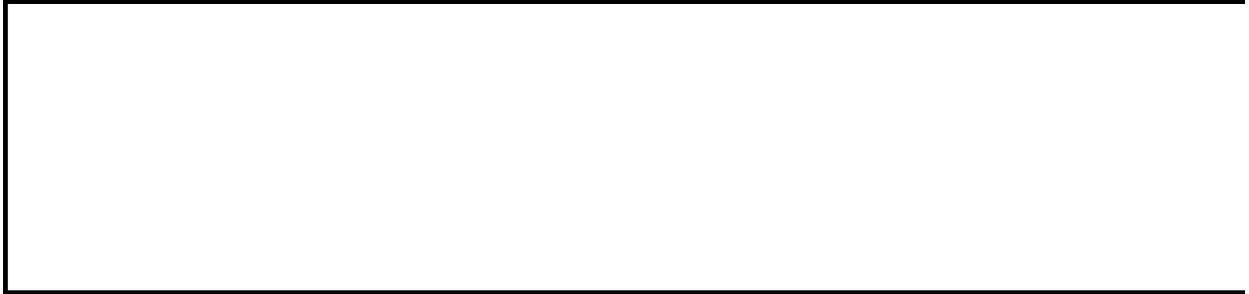
## Remarks

For the **Font** object, this property has no effect in Microsoft Windows, but its value is retained (it can be set and returned).

## Example

This example adds a shadow to the title of myChart.

```
Charts("Chart1").ChartTitle.Shadow = True
```



# Shape Property

Returns a **Shape** object that represents the shape attached to the specified comment, diagram node, or hyperlink.

*expression*.**Shape**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example selects comment two on the active sheet.

```
ActiveSheet.Comments(2).Shape.Select
```



# ShapeRange Property

Returns a [ShapeRange](#) object that represents the specified object or objects.  
Read-only.

## Example

This example creates a shape range that represents the embedded charts on worksheet one.

```
Set sr = Worksheets(1).ChartObjects.ShapeRange
```



# Shapes Property

Returns a [Shapes](#) object that represents all the shapes on the worksheet or chart sheet. Read-only.

For information about returning a single member of a collection, see [Returning an Object from a Collection](#).

## Example

This example adds a blue dashed line to worksheet one.

```
With Worksheets(1).Shapes.AddLine(10, 10, 250, 250).Line
    .DashStyle = msoLineDashDotDot
    .ForeColor.RGB = RGB(50, 0, 128)
End With
```



# SharedWorkspace Property

Returns a **SharedWorkspace** object that represents the Document Workspace in which a specified document is located. Read-only.

*expression*.**SharedWorkspace**

*expression* Required. An expression that returns one of the objects in the Applies To list.



# SharePointFormula Property

Returns a **String** representing the formula a calculated column. The formula is expressed in Excel syntax (US English locale, A1 notation). Read-only **String**.

*expression*.**SharePointFormula**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

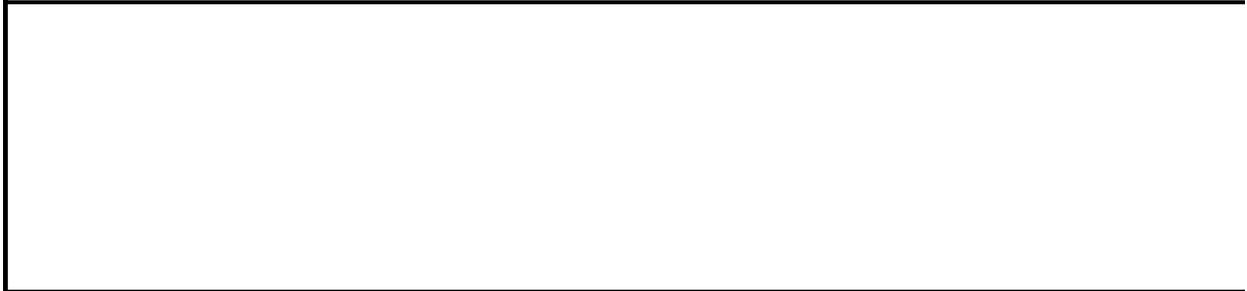
If the [ListColumn](#) object does not belong to a list that is linked to a SharePoint site or if it is not a column designated as a calculated column on the SharePoint site, you will get a run-time error.

## Example

The following example assumes that you have a list that contains four columns on a worksheet named "Sheet1" that is linked to a SharePoint site. The fourth column is a calculated column.

```
Dim wrksht As Worksheet
Dim oListObj As ListObject
Dim strSPFormula As String

Set wrksht = ActiveWorkbook.Worksheets("Sheet1")
Set oListObj = wrksht.ListObjects(1)
strSPFormula = oListObj.ListColumns(4).SharePointFormula
MsgBox (strSPFormula)
```



# SharePointURL Property

Returns a **String** representing the URL of the SharePoint list for a given **ListObject** object. Read-only **String**.

*expression*.**SharePointURL**

*expression* Required. An expression that returns a **ListObject** object.

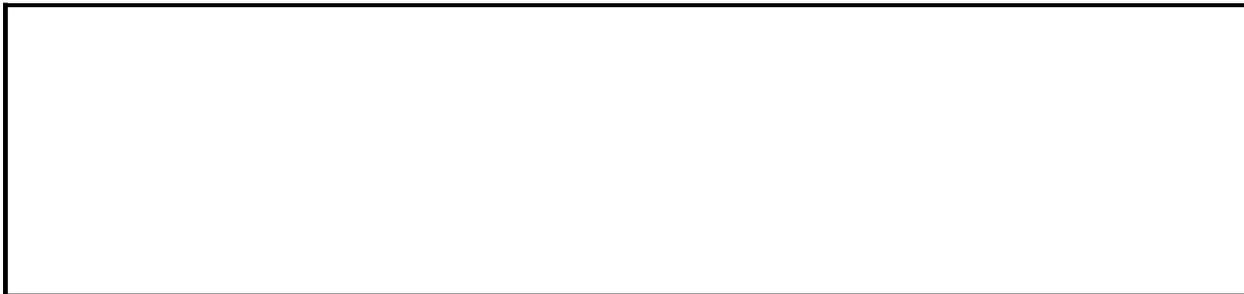
## Remarks

Accessing this property generates a run-time error if the list is not linked to a SharePoint site.

## Example

The following example sets elements of the *Target* parameter of the **Publish** method to push the **ListObject** object to a SharePoint site. The code sample uses the **SharePointURL** property to assign the URL to the array and the **Name** property to assign the name of the list. The information in the array is then passed to the SharePoint site using the **Publish** method.

```
Sub PublishList()  
    Dim wrksht As Worksheet  
    Dim objListObj As ListObject  
    Dim arTarget(4) As String  
    Dim strSTSCONNECTION As String  
  
    Set wrksht = ActiveWorkbook.Worksheets("Sheet1")  
    Set objListObj = wrksht.ListObjects(1)  
  
    arTarget(0) = "0"  
    arTarget(1) = objListObj.SharePointURL  
    arTarget(2) = "1"  
    arTarget(3) = objListObj.Name  
  
    strSTSCONNECTION = objListObj.Publish(arTarget, True)  
End Sub
```



# Sheet Property

Returns the sheet name for the specified [PublishObject](#) object. Read-only **String**.

## Example

This example determines the name of the worksheet that contains the first **PublishObject** object that is saved as static HTML in the Web page. The example then sets the **Boolean** variable `blnSheetFound` to **True**. If no items in the document have been saved as static HTML, `blnSheetFound` is **False**.

```
blnSheetFound = False
For Each objPO In Workbooks(1).PublishObjects
    If objPO.HtmlType = xlHTMLStatic Then
        strFirstPO = objPO.Sheet
        blnSheetFound = True
        Exit For
    End If
Next objPO
```



# Sheets Property

Returns a [Sheets](#) collection that represents all the sheets in the active workbook, for an **Application** object. Returns a **Sheets** collection that represents all the sheets in the specified workbook, for a **Workbook** object. Read-only **Sheets** object.

## Remarks

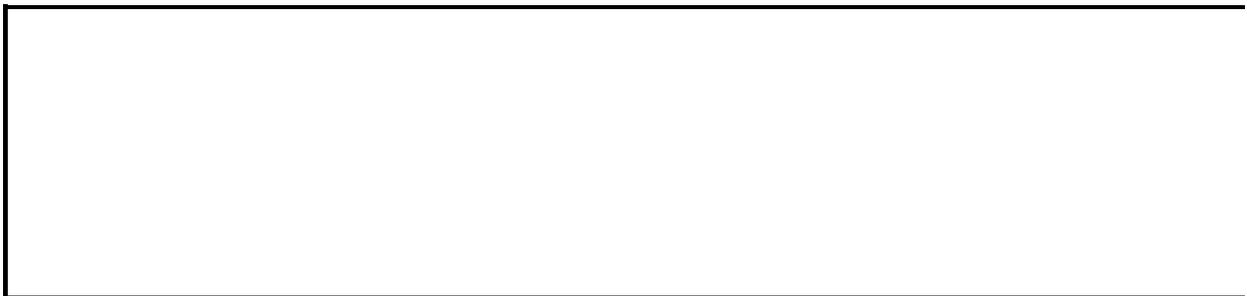
For information about returning a single member of a collection, see [Returning an Object from a Collection](#).

Using this property without an object qualifier is equivalent to using `ActiveWorkbook.Sheets`.

## Example

This example creates a new worksheet and then places a list of the active workbook's sheet names in the first column.

```
Set newSheet = Sheets.Add(Type:=xlWorksheet)
For i = 1 To Sheets.Count
    newSheet.Cells(i, 1).Value = Sheets(i).Name
Next i
```



# SheetsInNewWorkbook Property

Returns or sets the number of sheets that Microsoft Excel automatically inserts into new workbooks. Read/write **Long**.

## Example

This example displays the number of sheets automatically inserted into new workbooks.

```
MsgBox "Microsoft Excel inserts " & _  
    Application.SheetsInNewWorkbook & _  
    " sheet(s) in each new workbook"
```



# ShortcutKey Property

Returns or sets the shortcut key for a name defined as a custom Microsoft Excel 4.0 macro command. Read/write **String**.

## Example

This example sets the shortcut key for name one in the active workbook. The example should be run on a workbook in which name one refers to a Microsoft Excel 4.0 command macro.

```
ActiveWorkbook.Names(1).ShortcutKey = "K"
```



[Show All](#)

# ShowAllItems Property

**True** if all items in the PivotTable report are displayed, even if they don't contain summary data. The default value is **False**. Read/write **Boolean**.

## Remarks

For [OLAP](#) data sources, the value is always **False**.

## Example

This example displays all rows for the Month field in the first PivotTable report on worksheet one, including months for which there's no data.

```
Worksheets(1).PivotTables("Pivot1") _  
    .PivotFields("Month").ShowAllItems = True
```



# ShowAutoFilter Property

Returns **Boolean** to indicate whether the AutoFilter will be displayed.  
Read/write **Boolean**.

*expression*.**ShowAutoFilter**

*expression* Required. An expression that returns a **ListObject** object.

## Remarks

**ShowAutoFilter** property defaults to **True** for a new **ListObject** object.

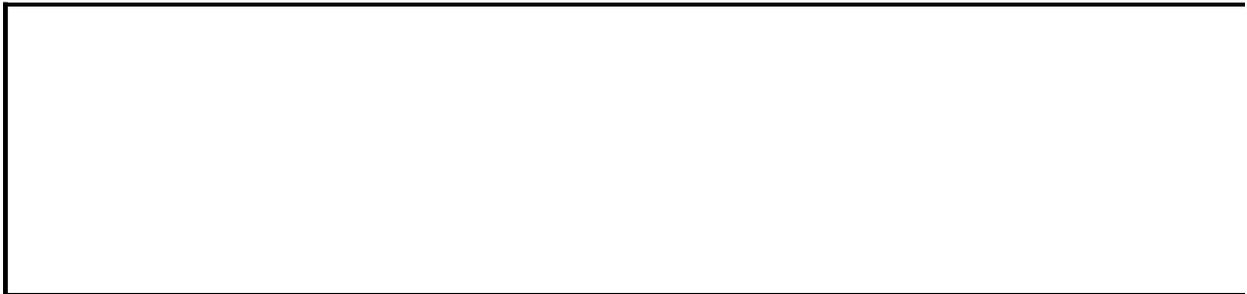
## Example

The following example displays the setting of the **ShowAutoFilter** property the default list in Sheet 1 of the active workbook.

```
Dim wrksht As Worksheet
Dim oListCol As ListColumn

Set wrksht = ActiveWorkbook.Worksheets("Sheet1")
Set oListCol = wrksht.ListObjects(1)

Debug.Print oListCol.ShowAutoFilter
```



# ShowBubbleSize Property

**True** to show the bubble size for the data labels on a chart. **False** to hide.  
Read/write **Boolean**.

*expression*.**ShowBubbleSize**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## **Remarks**

The chart must first be active before you can access the data labels programmatically or a run-time error will occur.

## Example

This example shows the bubble size for the data labels of the first series on the first chart. This example assumes a chart exists on the active worksheet.

```
Sub UseBubbleSize()  
    ActiveSheet.ChartObjects(1).Activate  
    ActiveChart.SeriesCollection(1) _  
        .DataLabels.ShowBubbleSize = True  
End Sub
```



# ShowCategoryName Property

**True** to display the category name for the data labels on a chart. **False** to hide.  
Read/write **Boolean**.

*expression*.**ShowCategoryName**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## **Remarks**

The chart must first be active before you can access the data labels programmatically or a run-time error will occur.

## Example

This example shows the category name for the data labels of the first series on the first chart. This example assumes a chart exists on the active worksheet.

```
Sub UseCategoryName()  
    ActiveSheet.ChartObjects(1).Activate  
    ActiveChart.SeriesCollection(1) _  
        .DataLabels.ShowCategoryName = True  
End Sub
```



# ShowChartTipNames Property

**True** if charts show chart tip names. The default value is **True**. Read/write **Boolean**.

## Example

This example turns off chart tip names and values.

```
With Application
    .ShowChartTipNames = False
    .ShowChartTipValue = False
End With
```



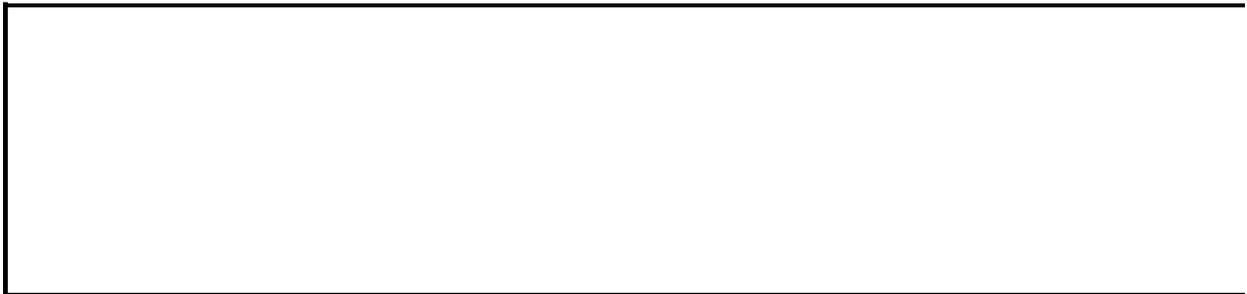
# ShowChartTipValues Property

**True** if charts show chart tip values. The default value is **True**. Read/write **Boolean**.

## Example

This example turns off chart tip names and values.

```
With Application
    .ShowChartTipNames = False
    .ShowChartTipValue = False
End With
```



# ShowConflictHistory Property

**True** if the Conflict History worksheet is visible in the workbook that's open as a shared list. Read/write **Boolean**.

## Remarks

If the specified workbook isn't open as a shared list, this property fails. To determine whether a workbook is open as a shared list, use the **MultiUserEditing** property.

## Example

This example determines whether the active workbook is open as a shared list. If it is, the example displays the Conflict History worksheet.

```
If ActiveWorkbook.MultiUserEditing Then  
    ActiveWorkbook.ShowConflictHistory = True  
End If
```



[Show All](#)

# ShowDetail Property

**True** if the outline is expanded for the specified range (so that the detail of the column or row is visible). The specified range must be a single summary column or row in an outline. Read/write **Variant**.

For the **PivotItem** object (or the **Range** object if the range is in a PivotTable report), this property is set to **True** if the item is showing detail.

## Remarks

This property isn't available for [OLAP](#) data sources.

If the specified range isn't in a PivotTable report, the following statements are true:

- The range must be in a single summary row or column.
- This property returns **False** if *any* of the children of the row or column are hidden.
- Setting this property to **True** is equivalent to unhiding all the children of the summary row or column.
- Setting this property to **False** is equivalent to hiding all the children of the summary row or column.

If the specified range is in a PivotTable report, it's possible to set this property for more than one cell at a time if the range is contiguous. This property can be returned only if the range is a single cell.

## Example

This example shows detail for the summary row of an outline on Sheet1. Before running this example, create a simple outline that contains a single summary row, and then collapse the outline so that only the summary row is showing. Select one of the cells in the summary row, and then run the example.

```
Worksheets("Sheet1").Activate  
Set myRange = ActiveCell.CurrentRegion  
lastRow = myRange.Rows.Count  
myRange.Rows(lastRow).ShowDetail = True
```



# ShowError Property

**True** if the data validation error message will be displayed whenever the user enters invalid data. Read/write **Boolean**.

## Example

This example adds data validation to cell A10 on worksheet one. The input value must be from 5 through 10; if the user types invalid data, an error message is displayed but no input message is displayed.

```
With Worksheets(1).Range("A10").Validation
    .Add Type:=xlValidateWholeNumber, _
        AlertStyle:=xlValidAlertStop, _
        Operator:=xlBetween, Formula1:="5", _
        Formula2:="10"
    .ErrorMessage = "value must be between 5 and 10"
    .ShowInput = False
    .ShowError = True
End With
```



# ShowImportExportValidationErrors Property

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Returns or sets whether to display a dialog box that details schema-validation errors when data is imported or exported through the specified XML schema map. The default value is **False**. Read/write **Boolean**.

*expression*.**ShowImportExportValidationErrors**

*expression* Required. An expression that returns an [XmlMap](#) object.



# ShowInFieldList Property

When set to **True** (default), a **CubeField** object will be shown in the field list.  
Read/write **Boolean**.

*expression*.**ShowInFieldList**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel determines if a **CubeField** object can be shown in the Field list and notifies the user. This example assumes a PivotTable report exists on the active worksheet and a **CubeField** object exists.

```
Sub IsCubeFieldInList()  
  
    Dim pvtTable As PivotTable  
    Dim cbeField As CubeField  
  
    Set pvtTable = ActiveSheet.PivotTables(1)  
    Set cbeField = pvtTable.CubeFields("[Country]")  
  
    ' Determine if a CubeField can be seen.  
    If cbeField.ShowInFieldList = True Then  
        MsgBox "The CubeField object can be seen in the field list."  
    Else  
        MsgBox "The CubeField object cannot be seen in the field list."  
    End If  
  
End Sub
```



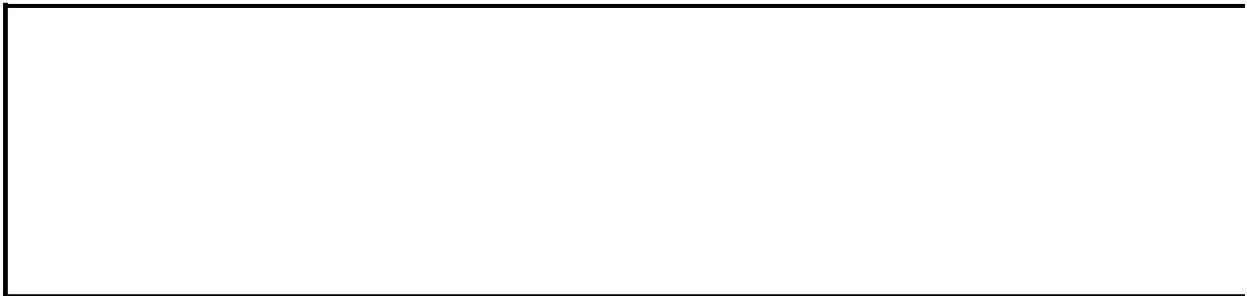
# ShowInput Property

**True** if the data validation input message will be displayed whenever the user selects a cell in the data validation range. Read/write **Boolean**.

## Example

This example adds data validation to cell A10. The input value must be from 5 through 10; if the user types invalid data, an error message is displayed but no input message is displayed.

```
With Worksheets(1).Range("A10").Validation
    .Add Type:=xlValidateWholeNumber, _
        AlertStyle:=xlValidAlertStop, _
        Operator:=xlBetween, Formula1:="5", _
        Formula2:="10"
    .ErrorMessage = "value must be between 5 and 10"
    .ShowInput = False
    .ShowError = True
End With
```



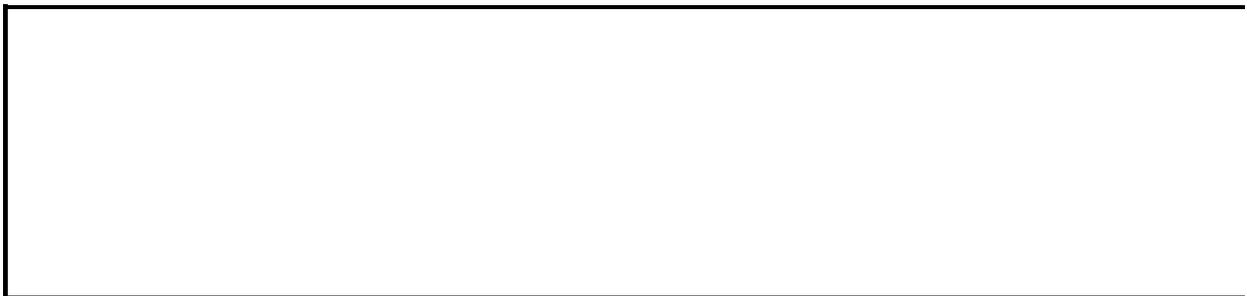
# ShowLegendKey Property

**True** if the data label legend key is visible. Read/write **Boolean**.

## Example

This example sets the data labels for series one in Chart1 to show values and the legend key.

```
With Charts("Chart1").SeriesCollection(1).DataLabels  
    .ShowLegendKey = True  
    .Type = xlShowValue  
End With
```



# ShowNegativeBubbles Property

**True** if negative bubbles are shown for the chart group. Valid only for bubble charts. Read/write **Boolean**.

## Example

This example makes negative bubbles visible for chart group one.

```
Worksheets(1).ChartObjects(1).Chart _  
    .ChartGroups(1).ShowNegativeBubbles = True
```



# ShowPageMultipleItemLabel Property

When set to **True** (default), "(Multiple Items)" will appear in the PivotTable cell on the worksheet whenever items are hidden and an aggregate of non-hidden items is shown in the PivotTable view. Read/write **Boolean**.

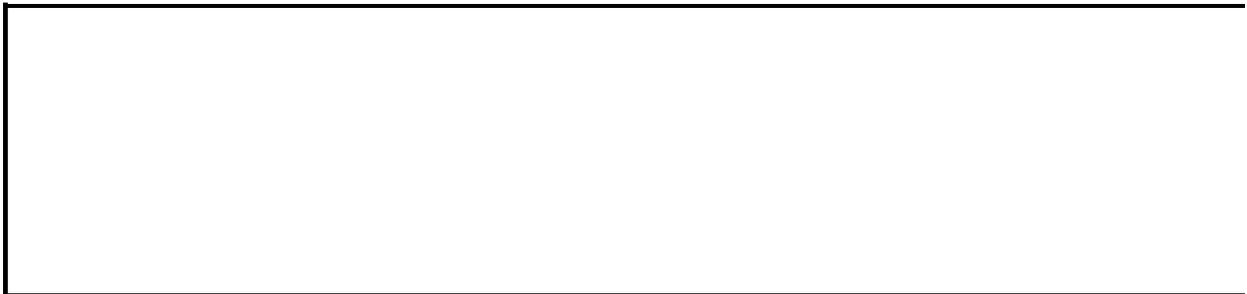
*expression*.**ShowPageMultipleItemLabel**

*expression* Required. An expression that returns a [PivotTable](#) object.

## Example

This example determines if "(Multiple Items)" will be displayed in the PivotTable cell and notifies the user. The example assumes that a PivotTable exists on the active worksheet.

```
Sub UseShowPageMultipleItemLabel()  
  
    Dim pvtTable As PivotTable  
  
    Set pvtTable = ActiveSheet.PivotTables(1)  
  
    ' Determine if multiple items are allowed.  
    If pvtTable.ShowPageMultipleItemLabel = True Then  
        MsgBox "The words 'Multiple Items' can be displayed."  
    Else  
        MsgBox "The words 'Multiple Items' cannot be displayed."  
    End If  
  
End Sub
```



# ShowPercentage Property

**True** to display the percentage value for the data labels on a chart. **False** to hide.  
Read/write **Boolean**.

*expression*.**ShowPercentage**

*expression* Required. An expression that returns one of the objects in the Applies To list.

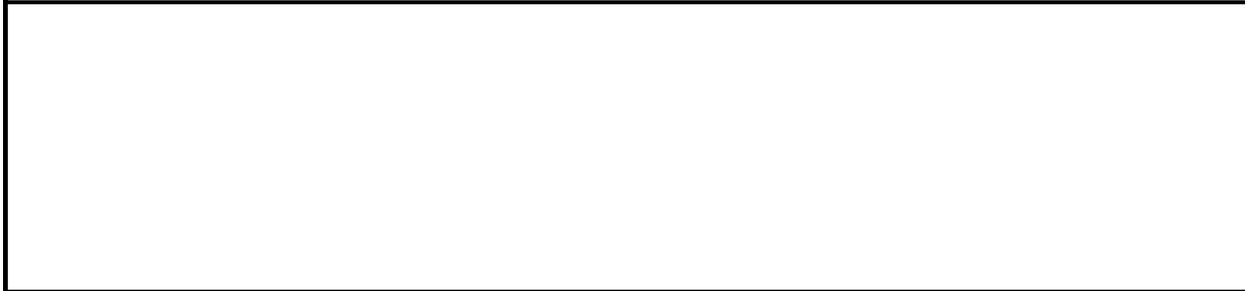
## Remarks

The chart must first be active before you can access the data labels programmatically or a run-time error will occur.

## Example

This example enables the percentage value to be shown for the data labels of the first series on the first chart. This example assumes a chart exists on the active worksheet.

```
Sub UsePercentage()  
  
    ActiveSheet.ChartObjects(1).Activate  
    ActiveChart.SeriesCollection(1) _  
        .DataLabels.ShowPercentage = True  
  
End Sub
```



# ShowPivotTableFieldList Property

**True** (default) if the PivotTable field list can be shown. Read/write **Boolean**.

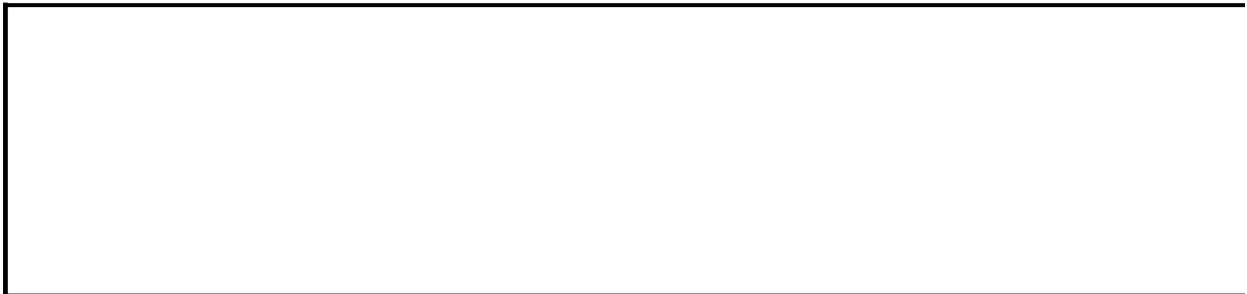
*expression*.**ShowPivotTableFieldList**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel determines if the PivotTable field list can be shown and notifies the user.

```
Sub UseShowPivotTableFieldList()  
    Dim wkbOne As Workbook  
  
    Set wkbOne = Application.ActiveWorkbook  
  
    'Determine PivotTable field list setting.  
    If wkbOne.ShowPivotTableFieldList = True Then  
        MsgBox "The PivotTable field list can be viewed."  
    Else  
        MsgBox "The PivotTable field list cannot be viewed."  
    End If  
  
End Sub
```



# ShowSeriesName Property

Returns or sets a **Boolean** to indicate the series name display behavior for the data labels on a chart. **True** to show the series name. **False** to hide. Read/write.

*expression*.**ShowSeriesName**

*expression* Required. An expression that returns one of the objects in the Applies To list.

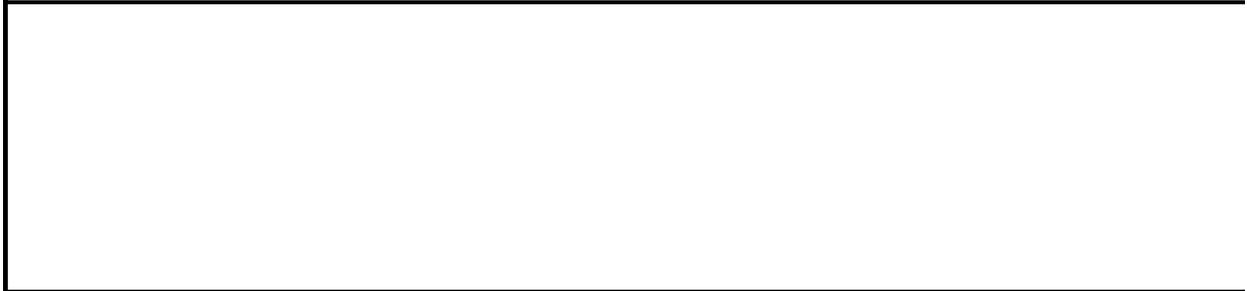
## **Remarks**

The chart must first be active before you can access the data labels programmatically or a run-time error will occur.

## Example

This example enables the series name to be shown for the data labels of the first series on the first chart. This example assumes a chart exists on the active worksheet.

```
Sub UseSeriesName()  
  
    ActiveSheet.ChartObjects(1).Activate  
    ActiveChart.SeriesCollection(1) _  
        .DataLabels.ShowSeriesName = True  
  
End Sub
```



# ShowStartupDialog Property

Returns **True** (default) when the New Workbook task pane appears for a Microsoft Excel application. Read/write **Boolean**.

*expression*.**ShowStartupDialog**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel determines if the New Workbook task pane appears and notifies the user.

```
Sub CheckStartupDialog()  
    ' Determine if the New Workbook task pane is enabled.  
    If Application.ShowStartupDialog = False Then  
        MsgBox "ShowStartupDialog is set to False."  
    Else  
        MsgBox "ShowStartupDialog is set to True."  
    End If  
End Sub
```



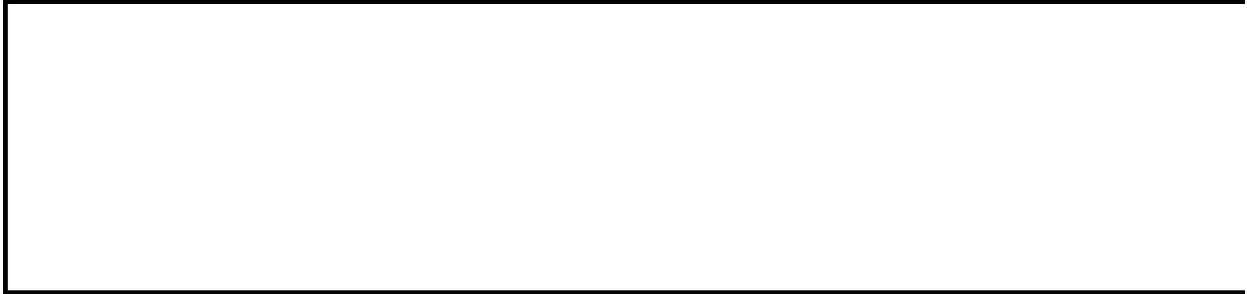
# ShowToolTips Property

**True** if ToolTips are turned on. Read/write **Boolean**.

## Example

This example causes Microsoft Excel to display ToolTips.

```
Application.ShowToolTips = True
```



# ShowTotals Property

Gets or sets a **Boolean** to indicate whether the Total row is visible. Read/write **Boolean**.

*expression*.**ShowTotals**

*expression* Required. An expression that returns **ListObject** object.

## Example

The following code sample displays the current setting of the **ShowTotals** property of the default list in Sheet1 of the active workbook.

```
Sub test()  
Dim oListObj As ListObject  
  
    Set wrksht = ActiveWorkbook.Worksheets("Sheet1")  
    Set oListObj = wrksht.ListObjects(1)  
  
    Debug.Print oListObj.ShowTotals  
End Sub
```



# ShowValue Property

Returns or sets a **Boolean** corresponding to a specified chart's data label values display behavior. **True** displays the values. **False** to hide. Read/write.

*expression*.**ShowValue**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## **Remarks**

The specified chart must first be active before you can access the data labels programmatically or a run-time error will occur.

## Example

This example enables the value to be shown for the data labels of the first series, on the first chart. This example assumes a chart exists on the active worksheet.

```
Sub UseValue()  
    ActiveSheet.ChartObjects(1).Activate  
    ActiveChart.SeriesCollection(1) _  
        .DataLabels.ShowValue = True  
End Sub
```



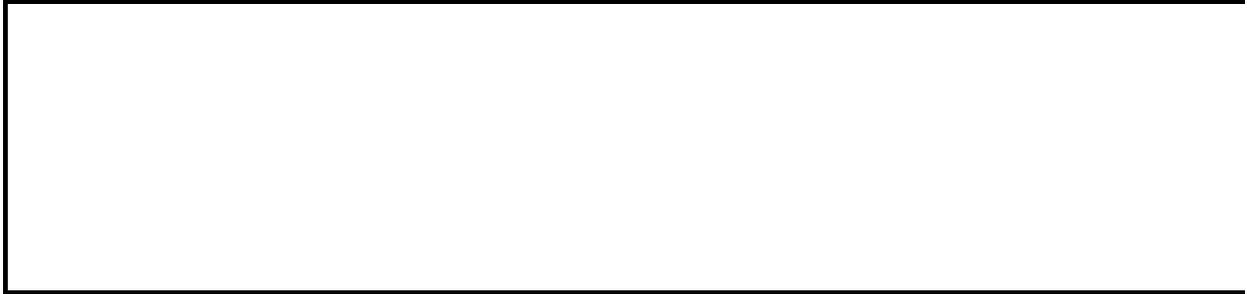
# ShowWindow Property

**True** if the embedded chart is displayed in a separate window. The **Chart** object used with this property must refer to an embedded chart. Read/write **Boolean**.

## Example

This example causes the embedded chart to be displayed in a separate window.

```
Worksheets(1).ChartObjects(1).Chart.ShowWindow = True
```



# ShowWindowsInTaskbar Property

**True** if there's a separate Windows taskbar button for each open workbook. The default value is **True**. Read/write **Boolean**.

*expression*.**ShowWindowsInTaskbar**

*expression* Required. An expression that returns one of the objects in the Applies To list.

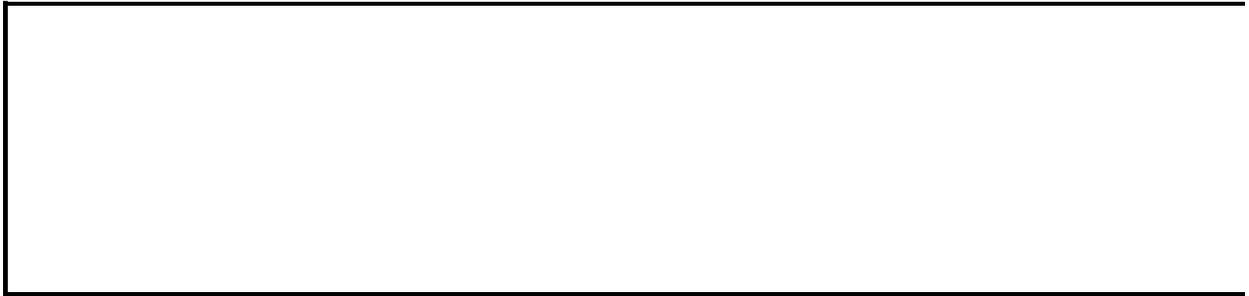
## Remarks

When set to **True**, this property simulates the look of a single-document interface (SDI), which makes it easier to navigate among open workbooks. However, if you work with multiple workbooks while other applications are open, you may want to set this property to **False** to avoid filling your taskbar with unnecessary buttons.

## Example

This example specifies that each open workbook won't have a separate Windows taskbar button.

```
Application.ShowWindowsInTaskbar = False
```



[Show All](#)

# ShrinkToFit Property

 [ShrinkToFit property as it applies to the \*\*Style\*\* object.](#)

**True** if text automatically shrinks to fit in the available column width.  
Read/write **Boolean**.

*expression*.**ShrinkToFit**

*expression* Required. An expression that returns a **Style** object.

 [ShrinkToFit property as it applies to the \*\*CellFormat\*\* and \*\*Range\*\* objects.](#)

**True** if text automatically shrinks to fit in the available column width. Returns **Null** if this property isn't set to the same value for all cells in the specified range.  
Read/write **Variant**.

*expression*.**ShrinkToFit**

*expression* Required. An expression that returns one of the above objects.

## Example

This example causes text in row one to automatically shrink to fit in the available column width.

```
Rows(1).ShrinkToFit = True
```



# Size Property

Returns or sets the size of the font. Read/write **Variant**.

## Example

This example sets the font size for cells A1:D10 on Sheet1 to 12 points.

```
With Worksheets("Sheet1").Range("A1:D10")  
    .Value = "Test"  
    .Font.Size = 12  
End With
```



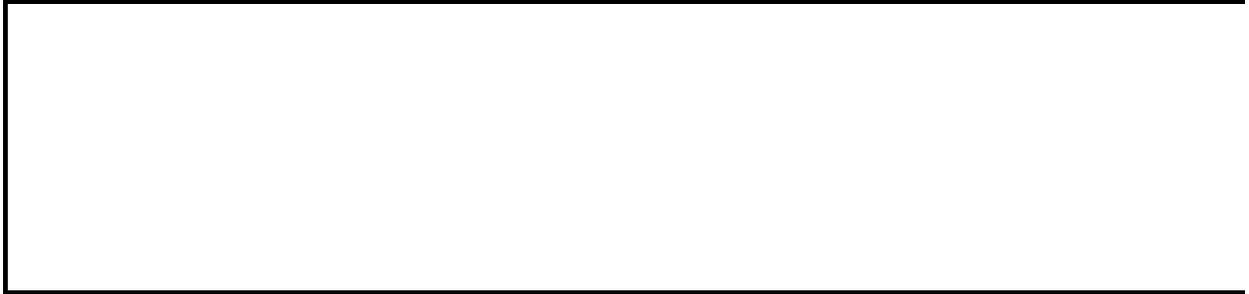
# SizeRepresents Property

Returns or sets what the bubble size represents on a bubble chart. Can be either of the following **XlSizeRepresents** constants: **xlSizeIsArea** or **xlSizeIsWidth**. Read/write **Long**.

## Example

This example sets what the bubble size represents for chart group one.

```
Charts(1).ChartGroups(1).SizeRepresents = xlSizeIsWidth
```



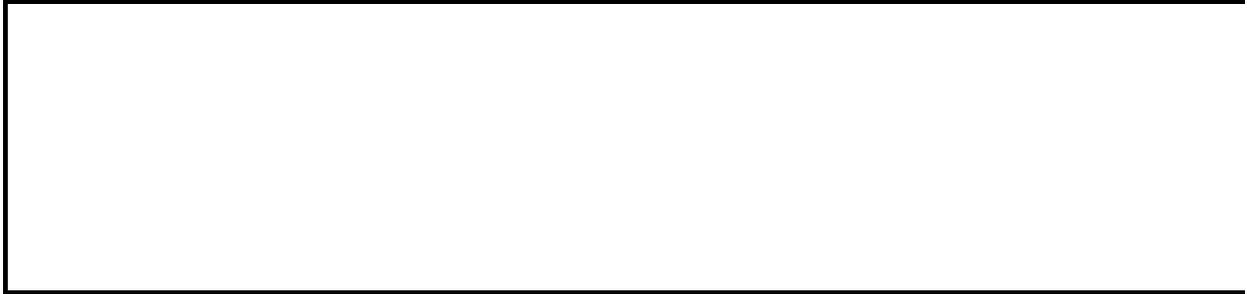
# SizeWithWindow Property

**True** if Microsoft Excel resizes the chart to match the size of the chart sheet window. **False** if the chart size isn't attached to the window size. Applies only to chart sheets (doesn't apply to embedded charts). Read/write **Boolean**.

## Example

This example sets Chart1 to be sized to its window.

```
Charts("Chart1").SizeWithWindow = True
```



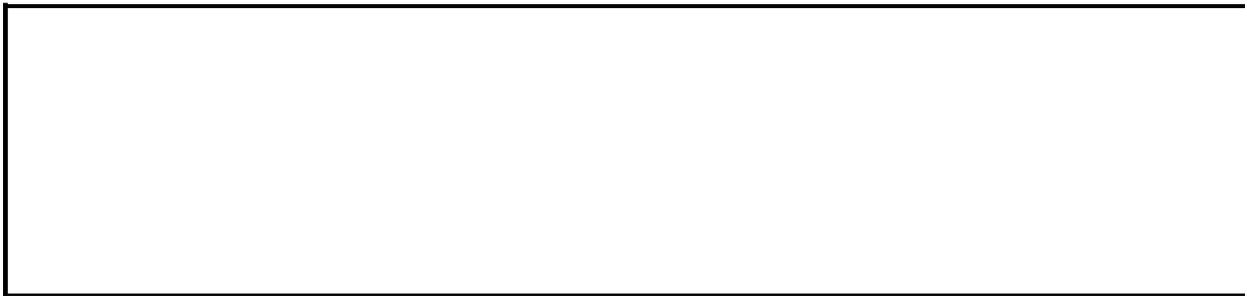
# SmallChange Property

Returns or sets the amount that the scroll bar or spinner is incremented or decremented for a line scroll (when the user clicks an arrow). Read/write **Long**.

## Example

This example creates a scroll bar and sets its linked cell, minimum, maximum, large change, and small change values.

```
Set sb = Worksheets(1).Shapes.AddFormControl(xlScrollBar, _  
    Left:=10, Top:=10, Width:=10, Height:=200)  
With sb.ControlFormat  
    .LinkedCell = "D1"  
    .Max = 100  
    .Min = 0  
    .LargeChange = 10  
    .SmallChange = 2  
End With
```



[Show All](#)

# SmallGrid Property

**True** if Microsoft Excel uses a grid that's two cells wide and two cells deep for a newly created PivotTable report. **False** if Excel uses a blank stencil outline.  
Read/write **Boolean**.

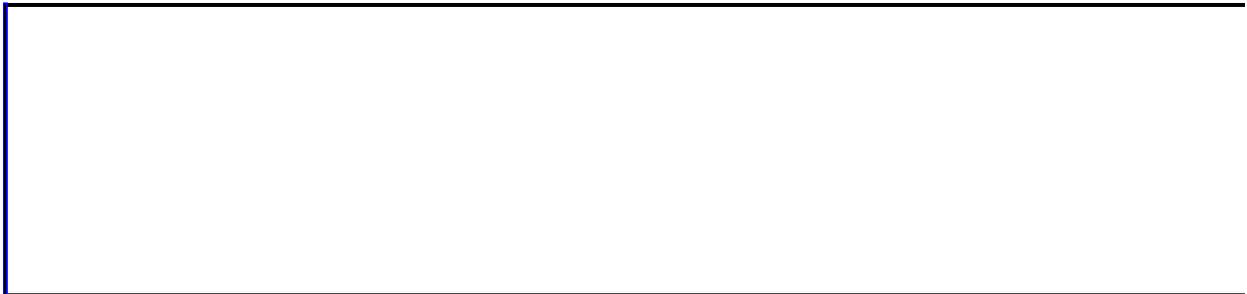
## Remarks

You should use the stencil outline. The grid is provided only because it enables compatibility with earlier versions of Excel.

## Example

This example creates a new PivotTable cache based on an [OLAP provider](#), and then it creates a new PivotTable report based on this cache, at cell A3 on the active worksheet. The example uses the stencil outline instead of the cell grid.

```
With ActiveWorkbook.PivotCaches.Add(SourceType:=xlExternal)
    .Connection = _
        "OLEDB;Provider=MSOLAP;Location=svrdata;Initial Catalog=Nati
    .MaintainConnection = True
    .CreatePivotTable TableDestination:=Range("A3"), _
        TableName:= "PivotTable1"
End With
With ActiveSheet.PivotTables("PivotTable1")
    .SmallGrid = False
    .PivotCache.RefreshPeriod = 0
    With .CubeFields("[state]")
        .Orientation = xlColumnField
        .Position = 0
    End With
    With .CubeFields("[Measures].[Count Of au_id]")
        .Orientation = xlDataField
        .Position = 0
    End With
End With
```



# SmartDocument Property

Returns a **SmartDocument** object that represents the settings for a smart document solution. Read-only.

*expression*.**SmartDocument**

*expression* Required. An expression that returns a [Workbook](#) object.

## Remarks

For more information on smart documents, please see the Smart Document Software Development Kit on the [Microsoft Developer Network \(MSDN\)](#) Web site.

---

# SmartTagActions Property

Returns a [SmartTagActions](#) object the type of action for a selected smart tag.

*expression*.**SmartTagActions**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## **Remarks**

An unrecognized smart tag action item will return a run-time error.

## Example

This example, Microsoft Excel places a smart tag titled "MSFT" in cell A1 and then notifies the user the smart tag action related to that smart tag. This example assumes the host system is connected to the Internet.

```
Sub UseSmartTagActions()  
    Dim strLink As String  
  
    strLink = "urn:schemas-microsoft-com:office:smartrtags#stockticke  
  
    ' Enable smart tags to be embedded and recognized.  
    ActiveWorkbook.SmartTagOptions.EmbedSmartTags = True  
    Application.SmartTagRecognizers.Recognize = True  
  
    Range("A1").Value = "MSFT"  
    MsgBox Range("A1").SmartTags.Add(strLink).SmartTagActions.Item(1  
End Sub
```



# SmartTagOptions Property

Returns a [SmartTagOptions](#) object representing the options that can be performed with a smart tag.

*expression*.**SmartTagOptions**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel notifies the user of the display settings for the smart tag options.

```
Sub CheckSmartTagOptions()  
    ' Check the display options for smart tags.  
    Select Case ActiveWorkbook.SmartTagOptions.DisplaySmartTags  
        Case xlButtonOnly  
            MsgBox "The button for smart tags will only be displayed  
        Case xlDisplayNone  
            MsgBox "Nothing will be displayed for smart tags."  
        Case xlIndicatorAndButton  
            MsgBox "The button and indicator will be displayed for s  
    End Select  
End Sub
```



# SmartTagRecognizers Property

Returns a [SmartTagRecognizers](#) collection for an application.

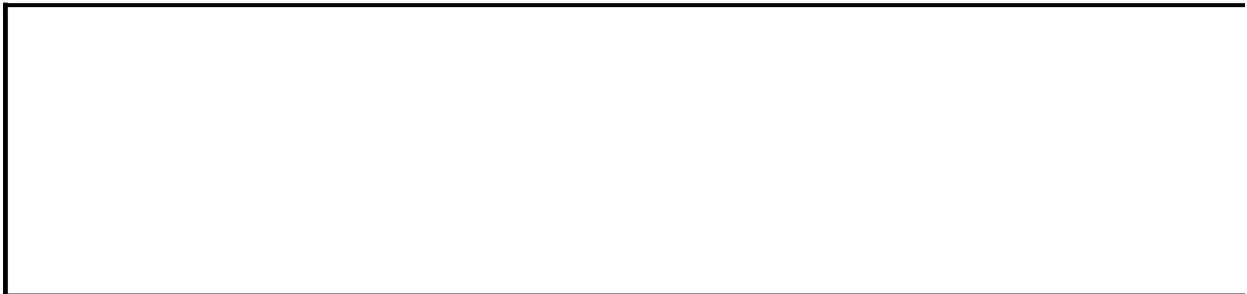
*expression*.**SmartTagRecognizers**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel displays the first smart tag recognizer item available for the application, or displays a message that none exist.

```
Sub CheckforSmartTagRecognizers()  
    ' Handle run-time error if no smart tag recognizers exist.  
    On Error Goto No_SmartTag_Recognizers_In_List  
  
    ' Notify the user of the first smart tag recognizer item.  
    MsgBox "The first smart tag recognizer is: " & _  
        Application.SmartTagRecognizers.Item(1)  
    Exit Sub  
  
No_SmartTag_Recognizers_In_List:  
    MsgBox "No smart tag recognizers exist in list."  
  
End Sub
```



# SmartTags Property

Returns a [SmartTags](#) object representing the identifier for the specified cell.

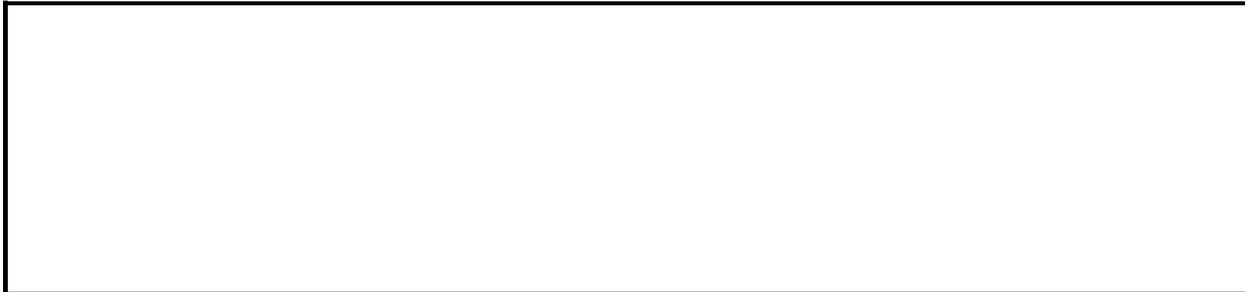
*expression*.**SmartTags**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example places a smart tag in cell A1 then notifies the user the parent of the identifier for cell A1, which is "MSFT". This example assumes the host system is connected to the Internet.

```
Sub ReturnSmartTag()  
  
    Dim strLink As String  
    Dim strType As String  
  
    ' Define SmartTag variables.  
    strLink = "urn:schemas-microsoft-com:smarctags#StockTickerSymbol  
    strType = "stockview"  
  
    ' Enable smart tags to be embedded and recognized.  
    ActiveWorkbook.SmartTagOptions.EmbedSmartTags = True  
    Application.SmartTagRecognizers.Recognize = True  
  
    Range("A1").Formula = "MSFT"  
    MsgBox Range("A1").SmartTags.Parent  
  
End Sub
```



# Smooth Property

**True** if curve smoothing is turned on for the line chart or scatter chart. Applies only to line and scatter charts. Read/write.

## Example

This example turns on curve smoothing for series one in Chart1. The example should be run on a 2-D line chart.

```
Charts("Chart1").SeriesCollection(1).Smooth = True
```



# SolveOrder Property

Returns a **Long** specifying the value of the calculated member's solve order MDX (Multidimensional Expression) argument. The default value is zero. Read-only.

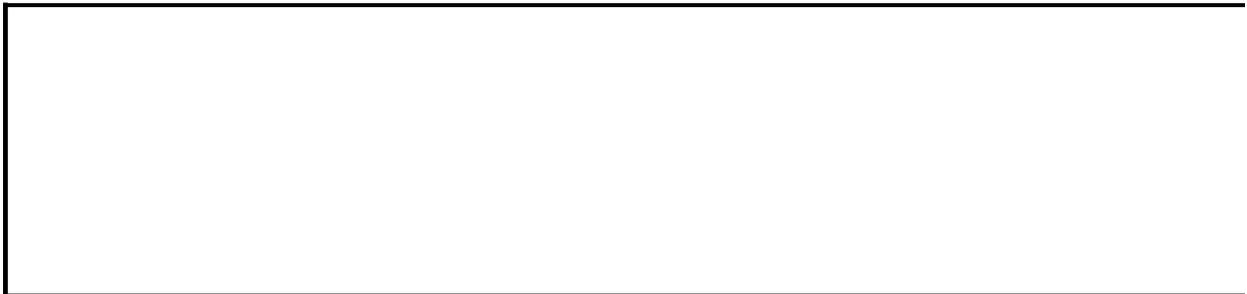
*expression*.**SolveOrder**

*expression* Required. An expression that returns a [CalculatedMember](#) object.

## Example

This example determines the solve order for a calculated member and notifies the user. The example assumes a PivotTable exists on the active worksheet.

```
Sub CheckSolveOrder()  
    Dim pvtTable As PivotTable  
  
    Set pvtTable = ActiveSheet.PivotTables(1)  
  
    ' Determine solve order and notify user.  
    If pvtTable.CalculatedMembers.Item(1).SolveOrder = 0 Then  
        MsgBox "The solve order is set to the default value."  
    Else  
        MsgBox "The solve order is not set to the default value."  
    End If  
  
End Sub
```



# SoundNote Property

This property should not be used. Sound notes have been removed from Microsoft Excel.



[Show All](#)

# Source Property

 [Source property as it applies to the \*\*PublishObject\*\* object.](#)

Returns the unique name that identifies items that have a **SourceType** property value of **xlSourceRange**, **xlSourceChart**, **xlSourcePrintArea**, **xlSourceAutoFilter**, **xlSourcePivotTable**, or **xlSourceQuery**. If the **SourceType** property is set to **xlSourceRange**, this property returns a range, which can be a defined name. If the **SourceType** property is set to **xlSourceChart**, **xlSourcePivotTable**, or **xlSourceQuery**, this property returns the name of the object, such as a chart name, a PivotTable report name, or a query table name. Read-only **String**.

*expression*.**Source**

*expression* Required. An expression that returns a **PublishObject** object.

 [Source property as it applies to the \*\*Watch\*\* object.](#)

Returns the unique name that identifies items that have a **SourceType** property value of **xlSourceRange**, **xlSourceChart**, **xlSourcePrintArea**, **xlSourceAutoFilter**, **xlSourcePivotTable**, or **xlSourceQuery**. If the **SourceType** property is set to **xlSourceRange**, this property returns a range, which can be a defined name. If the **SourceType** property is set to **xlSourceChart**, **xlSourcePivotTable**, or **xlSourceQuery**, this property returns the name of the object, such as a chart name, a PivotTable report name, or a query table name. Read-only **Variant**.

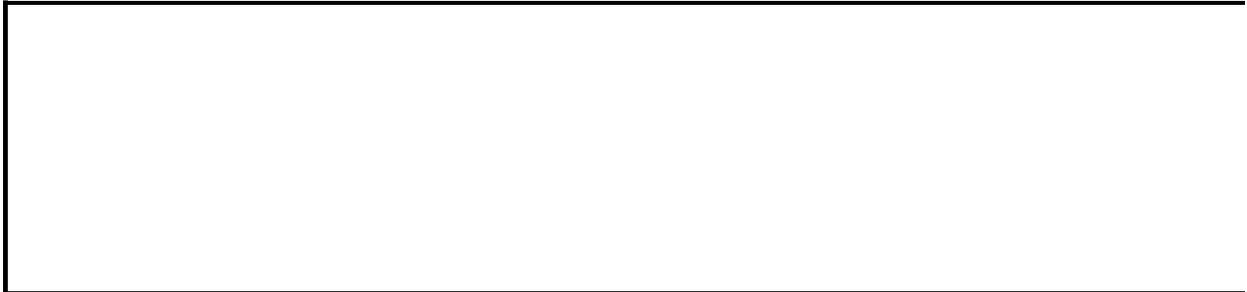
*expression*.**Source**

*expression* Required. An expression that returns a **Watch** object.

## Example

This example determines the unique name of the first chart (in the first workbook) saved as a Web page. and then it sets the **Boolean** variable `blnChartFound` to **True**. If no items in the document have been saved as Chart components, `blnChartFound` is **False**.

```
blnChartFound = False
For Each objP0 In Workbooks(1).PublishObjects
    If objP0.SourceType = xlSourceChart Then
        strFirstP0 = objP0.Source
        blnChartFound = True
        Exit For
    End If
Next objP0
```



# SourceConnectionFile Property

Returns or sets a **String** indicating the Microsoft Office Data Connection file or similar file that was used to create the PivotTable. Read/write.

*expression*.**SourceConnectionFile**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example determines if a connection exists for the PivotTable cache and, if there is a connection, displays the file name. If no connection exists, the code handles the run-time error and notifies the user. This example assumes a PivotTable exists on the active worksheet.

```
Sub CheckSourceConnection()  
  
    Dim pvtCache As PivotCache  
  
    Set pvtCache = Application.ActiveWorkbook.PivotCaches.Item(1)  
  
    On Error GoTo No_Connection  
  
    MsgBox "The source connection is: " & pvtCache.SourceConnectionF  
    Exit Sub  
  
No_Connection:  
    MsgBox "PivotCache source can not be determined."  
  
End Sub
```



# SourceData Property

Returns the data source for the PivotTable report, as shown in the following table. Read-write **Variant**.

<b>Data source</b>	<b>Return value</b>
Microsoft Excel list or database	The cell reference, as text.
External data source	An array. Each row consists of an SQL connection string with the remaining elements as the query string, broken down into 255-character segments.
Multiple consolidation ranges	A two-dimensional array. Each row consists of a reference and its associated page field items.
Another PivotTable report	One of the above three kinds of information.

## Remarks

This property is not available for OLE DB data sources.

## Example

Assume that you used an external data source to create a PivotTable report on Sheet1. This example inserts the SQL connection string and query string into a new worksheet.

```
Set newSheet = ActiveWorkbook.Worksheets.Add  
sdArray = Worksheets("Sheet1").UsedRange.PivotTable.SourceData  
For i = LBound(sdArray) To UBound(sdArray)  
    newSheet.Cells(i, 1) = sdArray(i)  
Next i
```



[Show All](#)

# SourceDataFile Property

 [SourceDataFile property as it applies to the \*\*PivotCache\*\* object.](#)

Returns a **String** indicating the source data file for the cache of the PivotTable.

*expression*.**SourceDataFile**

*expression* Required. An expression that returns a [PivotCache](#) object.

 [SourceDataFile property as it applies to the \*\*QueryTable\*\* object.](#)

Returns or sets a **String** indicating the source data file for a query table.

*expression*.**SourceDataFile**

*expression* Required. An expression that returns a [QueryTable](#) object.

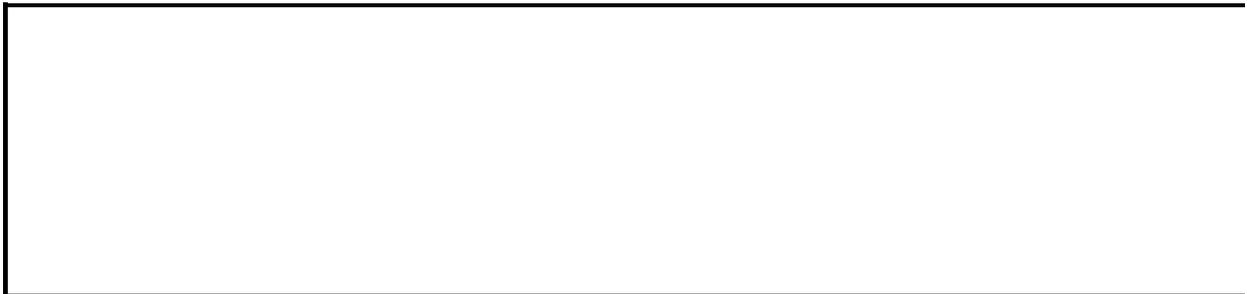
## Remarks

For file-based data sources (e.g. Access) the **SourceDataFile** property contains a fully qualified path to the source data file. It is null for server-based data sources (e.g. SQL Server). The **SourceDataFile** property is set to null if the [Connection](#) property is changed programmatically.

## Example

This example determines if a connection exists for the cache and, if there is a connection, displays the data source file name. If no connection exists, the code handles the run-time error and notifies the user. This example assumes a PivotTable exists on the active worksheet.

```
Sub CheckSourceConnection()  
    Dim pvtCache As PivotCache  
    Set pvtCache = Application.ActiveWorkbook.PivotCaches.Item(1)  
    On Error GoTo No_Connection  
    MsgBox "The data source connection is: " & _  
        pvtCache.SourceDataFile  
    Exit Sub  
No_Connection:  
    MsgBox "PivotCache source cannot be determined."  
End Sub
```



[Show All](#)

# SourceName Property

 [SourceName property as it applies to the \*\*OLEObject\*\* and \*\*OLEObjects\*\* objects.](#)

Returns or sets the specified object's link source name. Read/write **String**.

*expression*.**SourceName**

*expression* Required. An expression that returns one of the above objects.

 [SourceName property as it applies to the \*\*CalculatedMember\*\* and \*\*PivotField\*\* objects.](#)

Returns the specified object's name as it appears in the original source data for the specified PivotTable report. This might be different from the current item name if the user renamed the item after creating the PivotTable report. Read-only **String**.

*expression*.**SourceName**

*expression* Required. An expression that returns one of the above objects.

 [SourceName property as it applies to the \*\*PivotItem\*\* object.](#)

Returns the specified object's name as it appears in the original source data for the specified PivotTable report. This might be different from the current item name if the user renamed the item after creating the PivotTable report. Read-only **VARIANT**.

*expression*.**SourceName**

*expression* Required. An expression that returns a **PivotItem** object.

## Remarks

The following table shows example values of the **SourceName** property and related properties, given an [OLAP](#) data source with the unique name "[Europe].[France].[Paris]" and a non-OLAP data source with the item name "Paris".

<b>Property</b>	<b>Value (OLAP data source)</b>	<b>Value (non-OLAP data source)</b>
<b>Caption</b>	Paris	Paris
<b>Name</b>	[Europe].[France].[Paris] (read-only)	Paris
<b>SourceName</b>	[Europe].[France].[Paris] (read-only)	(same as SQL property value, read-only)
<b>Value</b>	[Europe].[France].[Paris] (read-only)	Paris

When specifying an index into the [PivotItems](#) collection, you can use the syntax shown in the following table.

<b>Syntax (OLAP data source)</b>	<b>Syntax (non-OLAP data source)</b>
expression.PivotItems("[Europe].[France].[Paris]")	expression.PivotItems("Paris")

When using the [Item](#) property to reference a specific member of a collection, you can use the text index names, as shown in the following table.

<b>Name (OLAP data source)</b>	<b>Name (non-OLAP data source)</b>
[Europe].[France].[Paris]	Paris

## Example

[As it applies to the \*\*PivotItem\*\* object.](#)

This example displays the original name (the name from the source database) of the item that contains the active cell.

```
Worksheets("Sheet1").Activate  
ActiveSheet.PivotTables(1).PivotSelect "1998", xlDataAndLabel  
MsgBox "The original item name is " & _  
    ActiveCell.PivotItem.SourceName
```



# SourceNameStandard Property

Returns a **String** that represents the PivotTable items' source name in standard English (United States) format settings. Read-only.

*expression*.**SourceNameStandard**

*expression* Required. An expression that returns one a [PivotItem](#) object.

## Remarks

This property is used when an item has a localized version and its **SourceNameStandard** property value differs from the [SourceName](#) property value, such as with date formatting.

## Example

This example displays the source name for the sixth item on the fifth field of the active PivotTable. The example assumes that a PivotTable exists on the active worksheet and that the data source contains at least five fields and six items per field.

```
Sub CheckSourceNameStandard()  
  
    Dim pvtTable As PivotTable  
    Dim pvtField As PivotField  
    Dim pvtItem As PivotItem  
  
    Set pvtTable = ActiveSheet.PivotTables(1)  
    Set pvtField = pvtTable.PivotFields(5)  
    Set pvtItem = pvtField.PivotItems(6)  
  
    ' Display source name.  
    MsgBox "The source name is: " & pvtItem.SourceNameStandard  
  
End Sub
```



# SourceRange Property

Returns a **Range** object that represents the cell that contains the value of the specified query parameter. Read-only.

## Example

This example changes the value of the cell used as the source range for the query.

```
Set qt = Sheets("sheet1").QueryTables(1)
Set param1 = qt.Parameters(1)
Set r = param1.SourceRange
r.Value = "New York"
qt.Refresh
```



[Show All](#)

# SourceType Property

 [SourceType property as it applies to the \*\*ListObject\*\* object.](#)

Returns a one of the [XlListObjectSourceType](#) constants indicating the current source of the list. Read-only.

XlListObjectSourceType can be one of these XlListObjectSourceType constants.

**xlSrcExternal**

**xlSrcRange**

**xlSrcXml**

*expression*.**SourceType**

*expression* Required. An expression that returns a **ListObject** object.

 [SourceType property as it applies to the \*\*PivotCache\*\* object.](#)

Returns a value that identifies the type of item being published. Read-only [XlPivotTableSourceType](#).

XlPivotTableSourceType can be one of these XlPivotTableSourceType constants.

**xlConsolidation**

**xlDatabase**

**xlExternal**

**xlPivotTable**

**xlScenario**

*expression*.**SourceType**

*expression* Required. An expression that returns a **PivotCache** object.

 [SourceType property as it applies to the \*\*PublishObject\*\* object.](#)

Returns a value that identifies the type of item being published. Read-only [XlSourceType](#).

XlSourceType can be one of these XlSourceType constants.

**xlSourceChart**

**xlSourcePrintArea**

**xlSourceRange**

**xlSourceWorkbook**

**xlSourceAutoFilter**

**xlSourcePivotTable**

**xlSourceQuery**

**xlSourceSheet**

*expression*.**SourceType**

*expression* Required. An expression that returns a **PublishObject** object.

## Example

This example determines the unique name of the first chart (in the first workbook) saved as a Web page, and then it sets the **Boolean** variable `blnChartFound` to **True**. If no items in the document have been saved as Chart components, `blnChartFound` is **False**.

```
blnChartFound = False
For Each objPO In Workbooks(1).PublishObjects
    If objPO.SourceType = xlSourceChart Then
        strFirstPO = objPO.Source
        blnChartFound = True
        Exit For
    End If
Next objPO
```

The following sample code returns a **XListObjectSourceType** constant indicating the source of the default list on Sheet1 of the active workbook.

```
Sub Test ()
    Dim wrksht As Worksheet
    Dim oListObj As ListObject

    Set wrksht = ActiveWorkbook.Worksheets("Sheet1")
    Set oListObj = wrksht.ListObjects(1)

    Debug.Print oListObj.SourceType
End Sub
```



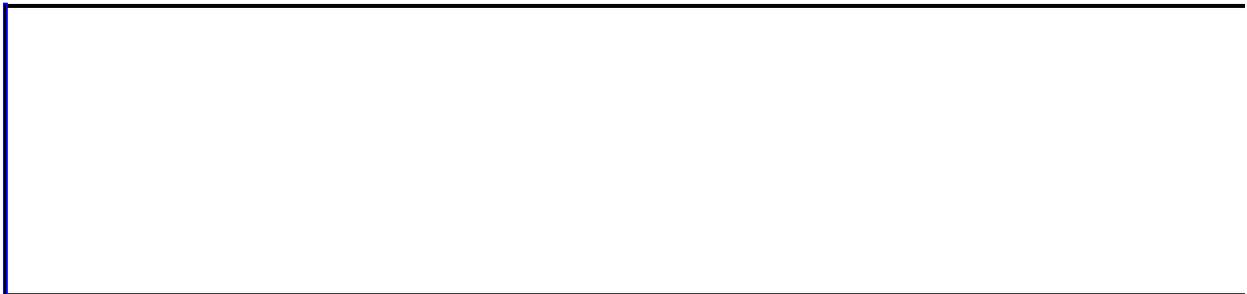
# SourceUrl Property

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Returns a **String** that represents the path to the XML data file, the Data Retrieval Service Connection (.uxdc) file, or the Web service that provides the source data for the specified data binding. Read-only.

*expression*.**SourceUrl**

*expression* Required. An expression that returns an [XmlDataBinding](#) object.



# SpeakCellOnEnter Property

Microsoft Excel supports a mode where the active cell will be spoken when the ENTER key is pressed or when the active cell is finished being edited. Setting the **SpeakCellOnEnter** property to **True** will turn this mode on. **False** turns this mode off. Read/write **Boolean**.

*expression*.**SpeakCellOnEnter**

*expression* Required. An expression that returns a **Speech** object.

## Example

This example determines if the active cell will be spoken when the ENTER key is pressed or the active cell is done being edited, and notifies the user.

```
Sub SpeechCheck()  
    ' Determine mode setting and notify user.  
    If Application.Speech.SpeakCellOnEnter = True Then  
        MsgBox "The Speak On Enter mode is turned on. " & _  
            "The active cell will be spoken when the ENTER "& _  
            "key is pressed or it is done being edited."  
    Else  
        MsgBox "The Speaker On Enter mode is turned off."  
    End If  
End Sub
```



# Speech Property

Returns a [Speech](#) object.

*expression*.**Speech**

*expression* Required. An expression that returns an **Application** object.

## Example

In the following example, Microsoft Excel plays back "Hello". This example assumes speech features have been installed on the host system.

```
Sub UseSpeech()  
    Application.Speech.Speak "Hello"  
End Sub
```

**Note** There is a speech feature in the setup tree that pertains to Dictation and Command & Control that does not have to be installed.



# SpellingOptions Property

Returns a [SpellingOptions](#) object that represents the spelling options of the application.

*expression*.**SpellingOptions**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel checks the setting on the spelling options for ignoring mixed digits, and notifies the user of its status.

```
Sub MixedDigitCheck()  
    ' Determine the setting on spell checking for mixed digits.  
    If Application.SpellingOptions.IgnoreMixedDigits = True Then  
        MsgBox "The spelling options are set to ignore mixed digits."  
    Else  
        MsgBox "The spelling options are set to check for mixed digi  
    End If  
End Sub
```



# Split Property

**True** if the window is split. Read/write **Boolean**.

## Remarks

It's possible for [FreezePanes](#) to be **True** and **Split** to be **False**, or vice versa.

This property applies only to worksheets and macro sheets.

## Example

This example splits the active window in Book1.xls at cell B2, without freezing panes. This causes the **Split** property to return **True**.

```
Workbooks("BOOK1.XLS").Worksheets("Sheet1").Activate
With ActiveWindow
    .SplitColumn = 2
    .SplitRow = 2
End With
```

This example illustrates two ways of removing the split added by the preceding example.

```
Workbooks("BOOK1.XLS").Worksheets("Sheet1").Activate
ActiveWindow.Split = False           'method one
Workbooks("BOOK1.XLS").Worksheets("Sheet1").Activate
ActiveWindow.SplitColumn = 0         'method two
ActiveWindow.SplitRow = 0
```

This example removes the window split. Before you can remove the split, you must set **FreezePanels** to **False** to remove frozen panes.

```
Workbooks("BOOK1.XLS").Worksheets("Sheet1").Activate
With ActiveWindow
    .FreezePanels = False
    .Split = False
End With
```



# SplitColumn Property

Returns or sets the column number where the window is split into panes (the number of columns to the left of the split line). Read/write **Long**.

## Example

This example splits the window and leaves 1.5 columns to the left of the split line.

```
Workbooks("BOOK1.XLS").Worksheets("Sheet1").Activate  
ActiveWindow.SplitColumn = 1.5
```



[Show All](#)

# SplitHorizontal Property

Returns or sets the location of the horizontal window split, in [points](#). Read/write **Double**.

## Example

This example sets the horizontal split for the active window to 216 points (3 inches).

```
Workbooks("BOOK1.XLS").Worksheets("Sheet1").Activate  
ActiveWindow.SplitHorizontal = 216
```



# SplitRow Property

Returns or sets the row number where the window is split into panes (the number of rows above the split). Read/write **Long**.

## Example

This example splits the active window so that there are 10 rows above the split line.

```
Workbooks("BOOK1.XLS").Worksheets("Sheet1").Activate  
ActiveWindow.SplitRow = 10
```



[Show All](#)

# SplitType Property

Returns or sets the way the two sections of either a pie of pie chart or a bar of pie chart are split. Read/write [XlChartSplitType](#).

XlChartSplitType can be one of these XlChartSplitType constants.

**xlSplitByCustomSplit**

**xlSplitByPercentValue**

**xlSplitByPosition**

**xlSplitByValue**

*expression*.**SplitType**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example must be run on either a pie of pie chart or a bar of pie chart. The example splits the two sections of the chart by value, combining all values under 10 in the primary pie and displaying them in the secondary section.

```
With Worksheets(1).ChartObjects(1).Chart.ChartGroups(1)  
    .SplitType = xlSplitByValue  
    .SplitValue = 10  
    .VaryByCategories = True  
End With
```



# SplitValue Property

Returns or sets the threshold value separating the two sections of either a pie of pie chart or a bar of pie chart. Read/write **Variant**.

## Example

This example must be run on either a pie of pie chart or a bar of pie chart. The example splits the two sections of the chart by value, combining all values under 10 in the primary pie and displaying them in the secondary section.

```
With Worksheets(1).ChartObjects(1).Chart.ChartGroups(1)  
    .SplitType = xlSplitByValue  
    .SplitValue = 10  
    .VaryByCategories = True  
End With
```



[Show All](#)

# SplitVertical Property

Returns or sets the location of the vertical window split, in [points](#). Read/write **Double**.

## Example

This example sets the vertical split for the active window to 216 points (3 inches).

```
Workbooks("BOOK1.XLS").Worksheets("Sheet1").Activate  
ActiveWindow.SplitVertical = 216
```



# SqlState Property

Returns the SQL state error. Read-only **String**.

## Remarks

For an explanation of the specific error, see you SQL documentation.

## Example

This example refreshes query table one and displays any ODBC errors that occur.

```
With Worksheets(1).QueryTables(1)
    .Refresh
    Set errs = Application.ODBCErrors
    If errs.Count > 0 Then
        Set r = .Destination.Cells(1)
        r.Value = "The following errors occurred:"
        c = 0
        For Each er In errs
            c = c + 1
            r.Offset(c, 0).Value = er.ErrorString
            r.Offset(c, 1).Value = er.SqlState
        Next
    Else
        MsgBox "Query complete: all records returned."
    End If
End With
```



# Stage Property

Returns a numeric value specifying the stage of an error that resulted after the most recent OLE DB query. Read-only **Long**.

## Example

This example displays the error numbers, stage, and other error information returned by the most recent OLE DB query.

```
Set objEr = Application.OLEDBErrors(1)
MsgBox "The following error occurred:" & _
    objEr.Number & ", " & objEr.Native & ", " & _
    objEr.Stage & ", " & _
    objEr.ErrorString & " : " & objEr.SqlState
```



# StandardFont Property

Returns or sets the name of the standard font. Read/write **String**.

## **Remarks**

If you change the standard font by using this property, the change doesn't take effect until you restart Microsoft Excel.

## Example

This example sets the standard font to Geneva (on the Macintosh) or Arial (in Windows).

```
If Application.OperatingSystem Like "*Macintosh*" Then
    Application.StandardFont = "Geneva"
Else
    Application.StandardFont = "Arial"
End If
```



[Show All](#)

# StandardFontSize Property

Returns or sets the standard font size, in [points](#). Read/write **Long**.

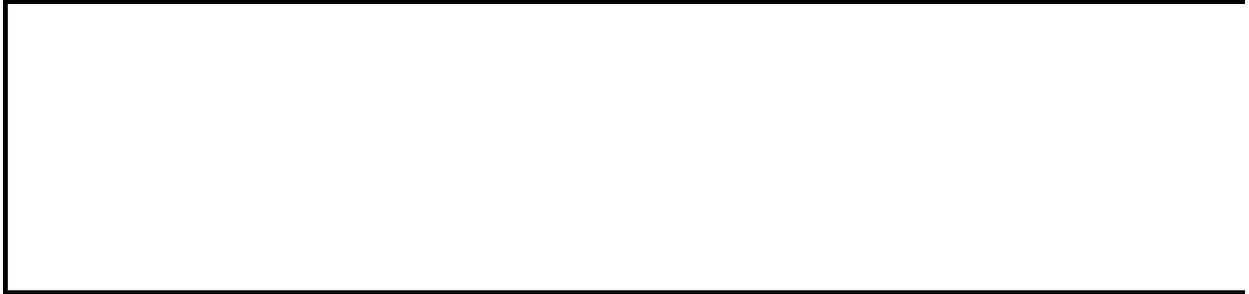
## **Remarks**

If you change the standard font size by using this property, the change doesn't take effect until you restart Microsoft Excel.

## Example

This example sets the standard font size to 12 points.

```
Application.StandardFontSize = 12
```



# StandardFormula Property

Returns or sets a **String** specifying formulas with standard English (United States) formatting. Read/write.

*expression*.**StandardFormula**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

The **StandardFormula** property primarily affects item names with date or number formatting. It provides a way to specify or query a formula for a given calculated item.

The **StandardFormula** property is "international-friendly" whereas the **Formula** property is not.

## Example

This example adds 10 to the Decimals field and displays it as a calculated item in the data field. The example assumes that a PivotTable exists on the active worksheet and that a field titled "Decimals" exists in the data table.

```
Sub UseStandardFomula()  
  
    Dim pvtTable As PivotTable  
    Set pvtTable = ActiveSheet.PivotTables(1)  
  
    ' Change calculated field of decimals by adding '10'.  
    pvtTable.CalculatedFields.Item(1).StandardFormula = "Decimals +  
End Sub
```



[Show All](#)

# StandardHeight Property

Returns the standard (default) height of all the rows in the worksheet, in [points](#).  
Read-only **Double**.

## Example

This example sets the height of row one on Sheet1 to the standard height.

```
Worksheets("Sheet1").Rows(1).RowHeight = _  
    Worksheets("Sheet1").StandardHeight
```



# StandardWidth Property

Returns or sets the standard (default) width of all the columns in the worksheet.  
Read/write **Double**.

## **Remarks**

One unit of column width is equal to the width of one character in the Normal style. For proportional fonts, the width of the character 0 (zero) is used.

## Example

This example sets the width of column one on Sheet1 to the standard width.

```
Worksheets("Sheet1").Columns(1).ColumnWidth = _  
    Worksheets("Sheet1").StandardWidth
```



# Start Property

Returns the position that represents the first character of a phonetic text string in the specified cell. Read-only **Long**.

## Example

This example returns the starting position of the second phonetic text string in the active cell.

```
ActiveCell.FormulaR1C1 = "東京都渋谷区代々木"  
ActiveCell.Phonetics.Add Start:=1, Length:=3, Text:="トウキョウト"  
ActiveCell.Phonetics.Add Start:=4, Length:=3, Text:="シブヤク"  
MsgBox ActiveCell.Phonetics(2).Start
```



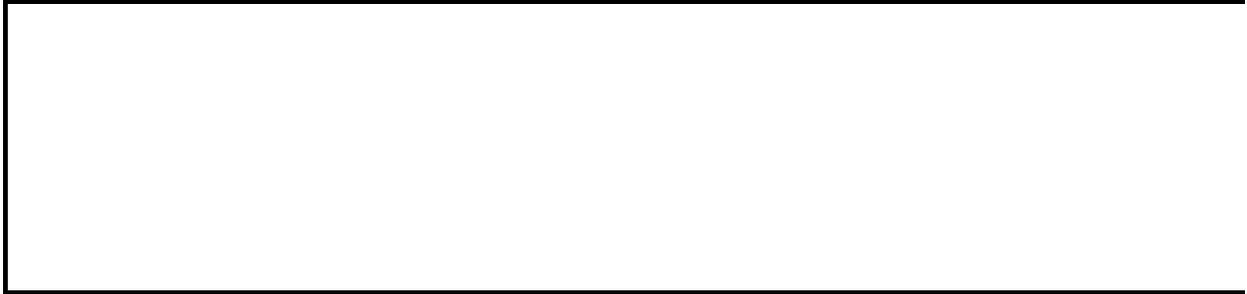
# StartupPath Property

Returns the complete path of the startup folder, excluding the final separator.  
Read-only **String**.

## Example

This example displays the full path to the Microsoft Excel startup folder.

```
MsgBox Application.StartupPath
```



[Show All](#)

# Status Property

Indicates the status of the routing slip. Read-only [XIRoutingSlipStatus](#).

XIRoutingSlipStatus can be one of these XIRoutingSlipStatus constants.

**xlNotYetRouted**

**xlRoutingComplete**

**xlRoutingInProgress**

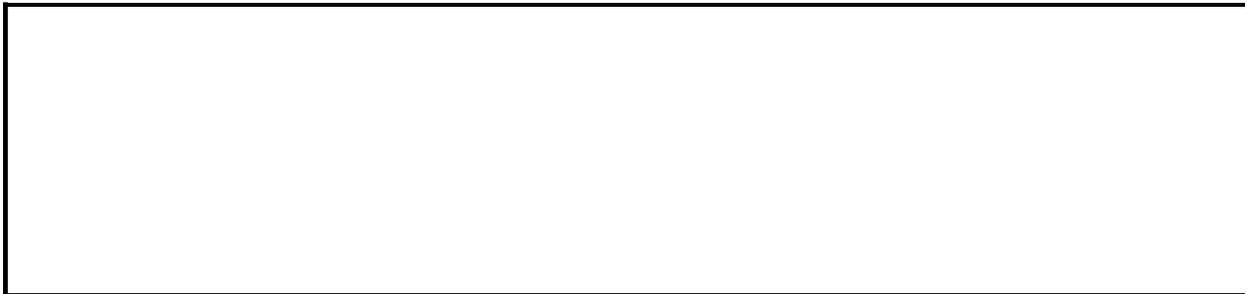
*expression*.**Status**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example resets the routing slip for Book1.xls if routing has been completed.

```
With Workbooks("BOOK1.XLS").RoutingSlip
    If .Status = xlRoutingComplete Then
        .Reset
    Else
        MsgBox "Cannot reset routing; not yet complete."
    End If
End With
```



# StatusBar Property

Returns or sets the text in the status bar. Read/write **String**.

## Remarks

This property returns **False** if Microsoft Excel has control of the status bar. To restore the default status bar text, set the property to **False**; this works even if the status bar is hidden.

## Example

This example sets the status bar text to "Please be patient..." before it opens the workbook Large.xls, and then it restores the default text.

```
oldStatusBar = Application.DisplayStatusBar
Application.DisplayStatusBar = True
Application.StatusBar = "Please be patient..."
Workbooks.Open filename:="LARGE.XLS"
Application.StatusBar = False
Application.DisplayStatusBar = oldStatusBar
```



# Strikethrough Property

**True** if the font is struck through with a horizontal line. Read/write **Boolean**.

## Example

This example sets the font in the active cell on Sheet1 to strikethrough.

```
Worksheets("Sheet1").Activate  
ActiveCell.Font.Strikethrough = True
```



[Show All](#)

# Style Property

[Style property as it applies to the \*\*LineFormat\*\* object.](#)

Returns a **Style** object that represents the style of the specified range. Read/write **MsoLineStyle**.

MsoLineStyle can be one of these MsoLineStyle constants.

**msoLineSingle**

**msoLineThickBetweenThin**

**msoLineThinThick**

**msoLineStyleMixed**

**msoLineThickThin**

**msoLineThinThin**

*expression*.**Style**

*expression* Required. An expression that returns one of the above objects.

[Style property as it applies to the \*\*Range\*\* object.](#)

Returns a **Style** object that represents the style of the specified range. Read/write **Variant**.

*expression*.**Style**

*expression* Required. An expression that returns one of the above objects.

## Example

[As it applies to the \*\*Range\*\* object.](#)

This example applies the Normal style to cell A1 on Sheet1.

```
Worksheets("Sheet1").Range("A1").Style.Name = "Normal"
```

If cell B4 on Sheet1 currently has the Normal style applied, this example applies the Percent style.

```
If Worksheets("Sheet1").Range("B4").Style.Name = "Normal" Then  
    Worksheets("Sheet1").Range("B4").Style.Name = "Percent"  
End If
```



# Styles Property

Returns a [Styles](#) collection that represents all the styles in the specified workbook. Read-only.

For information about returning a single member of a collection, see [Returning an Object from a Collection](#).

## Example

This example deletes the user-defined style "Stock Quote Style" from the active workbook.

```
ActiveWorkbook.Styles("Stock Quote Style").Delete
```



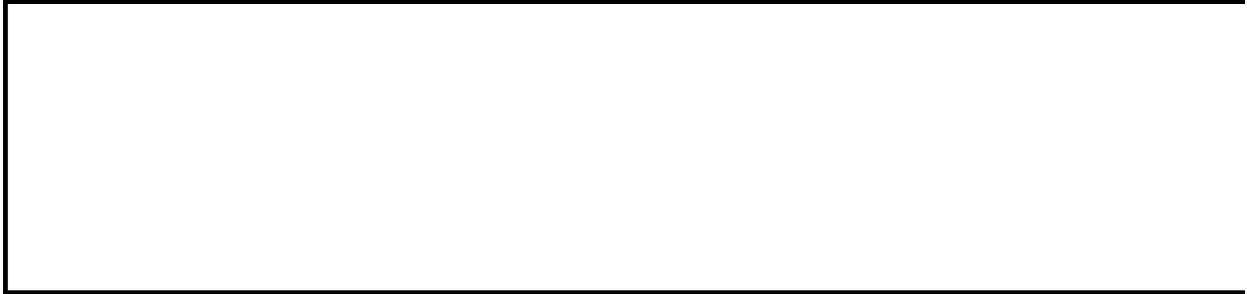
# SubAddress Property

Returns or sets the location within the document associated with the hyperlink.  
Read/write **String**.

## Example

This example topic adds a range location to the hyperlink for shape one.

```
worksheets(1).Shapes(1).Hyperlink.SubAddress = "A1:B10"
```



# Subject Property

Returns or sets the subject for the mailer or routing slip. Read/write **String**.

## Remarks

The subject for the **RoutingSlip** object is used as the subject for mail messages used to route the workbook.

## Example

This example sets the subject for a routing slip for the open workbook. To run this example, you must have Microsoft Exchange installed.

```
With ThisWorkbook
    .HasRoutingSlip = True
    With .RoutingSlip
        .Delivery = xlOneAfterAnother
        .Recipients = Array("Adam Bendel", _
            "Jean Selva", "Bernard Gabor")
        .Subject = "Here is the workbook"
        .Message = "Here is the workbook. What do you think?"
        .ReturnWhenDone = True
    End With
    .Route
End With
```



# Subscript Property

**True** if the font is formatted as subscript. **False** by default. Read/write **Variant**.

## Example

This example makes the second character in cell A1 a subscript character.

```
Worksheets("Sheet1").Range("A1") _  
    .Characters(2, 1).Font.Subscript = True
```



[Show All](#)

# SubtotalHiddenPageItems Property

**True** if hidden page field items in the PivotTable report are included in row and column subtotals, block totals, and grand totals. The default value is **False**.  
Read/write **Boolean**.

## Remarks

For [OLAP](#) data sources, the value is always **True**.

## Example

This example sets the first PivotTable report on worksheet one to exclude hidden page field items in subtotals.

```
Worksheets(1).PivotTables("Pivot1").SubtotalHiddenPageItems = True
```



# SubtotalName Property

Returns or sets the text string label displayed in the subtotal column or row heading in the specified PivotTable report. The default value is the string "Subtotal". Read/write **String**.

## Example

This example sets the subtotal label to "Regional Subtotal" (instead of the default string "Subtotal") in the state field in the second PivotTable report on the active worksheet.

```
ActiveSheet.PivotTables("PivotTable2") _  
    .PivotFields("state").SubtotalName = "Regional Subtotal"
```



[Show All](#)

# Subtotals Property

Returns or sets subtotals displayed with the specified field. Valid only for nondata fields. Read/write **Variant**.

*expression*.**Subtotals**(*Index*)

*expression* Required. An expression that returns a **PivotField** object.

**Index** Optional **Variant**. A subtotal index, as shown in the following table. If this argument is omitted, the **Subtotals** method returns an array that contains a Boolean value for each subtotal.

<b>Index</b>	<b>Meaning</b>
1	Automatic
2	Sum
3	Count
4	Average
5	Max
6	Min
7	Product
8	Count Nums
9	StdDev
10	StdDevp
11	Var
12	Varp

If an index is **True**, the field shows that subtotal. If index 1 (Automatic) is **True**, all other values are set to **False**.

## Remarks

For [OLAP](#) data sources, *Index* can only return or be set to 1 (Automatic). The returned array always contains **True** or **False** for the first array element, and it contains **False** for all other elements. An array of element values that are all **False** indicates that there are no subtotals.

## Example

This example sets the field that contains the active cell to show Sum subtotals.

```
Worksheets("Sheet1").Activate  
ActiveCell.PivotField.Subtotals(2) = True
```



# SuggestMainOnly Property

When set to **True**, instructs Microsoft Excel to suggest words from only the main dictionary, for using the spelling checker. **False** removes the limits of suggesting words from only the main dictionary, for using the spelling checker. Read/write **Boolean**.

*expression*.**SuggestMainOnly**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel checks the spell checking options for suggesting words only from the main dictionary and reports the status to the user.

```
Sub UsingMainDictionary()  
    ' Check the setting of suggesting words only from the main dicti  
    If Application.SpellingOptions.SuggestMainOnly = True Then  
        MsgBox "Spell checking option suggestions will only come fro  
    Else  
        MsgBox "Spell checking option suggestions are not limited to  
    End If  
End Sub
```



# Summary Property

**True** if the range is an outlining summary row or column. The range should be a row or a column. Read-only **Variant**.

## Example

This example formats row four on Sheet1 as bold and italic if it's an outlining summary column.

```
With Worksheets("Sheet1").Rows(4)
    If .Summary = True Then
        .Font.Bold = True
        .Font.Italic = True
    End If
End With
```



[Show All](#)

# SummaryColumn Property

Returns or sets the location of the summary columns in the outline. Read/write [XLSummaryColumn](#).

XLSummaryColumn can be one of these XLSummaryColumn constants.

**xlSummaryOnRight.** The summary column will be positioned to the right of the detail columns in the outline.

**xlSummaryOnLeft.** The summary column will be positioned to the left of the detail columns in the outline.

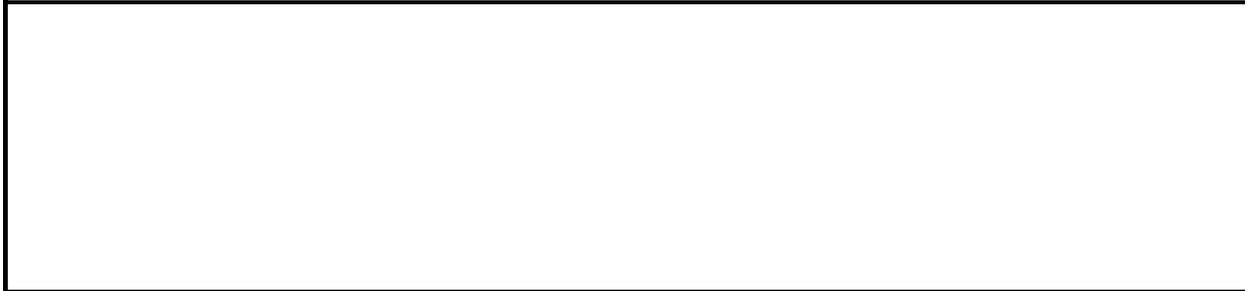
*expression*.**SummaryColumn**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example creates an outline with automatic styles, with the summary row above the detail rows, and with the summary column to the right of the detail columns.

```
Worksheets("Sheet1").Activate  
Selection.AutoOutline  
With ActiveSheet.Outline  
    .SummaryRow = xlAbove  
    .SummaryColumn = xlRight  
    .AutomaticStyles = True  
End With
```



[Show All](#)

# SummaryRow Property

Returns or sets the location of the summary rows in the outline. Read/write [XISummaryRow](#).

XISummaryRow can be one of these XISummaryRow constants.

**xlSummaryBelow.** The summary row will be positioned below the detail rows in the outline.

**xlSummaryAbove.** The summary row will be positioned above the detail rows in the outline.

*expression*.**SummaryRow**

*expression* Required. An expression that returns one of the objects in the Applies To list.

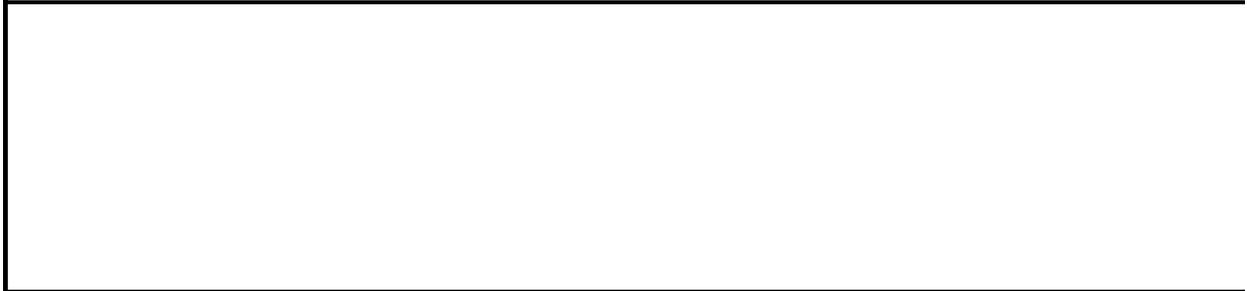
## Remarks

Set **SummaryRow** to **xlAbove** for Microsoft Word-style outlines, where category headers are above the detailed information. Set **SummaryRow** to **xlBelow** for accounting-style outlines, where summations are below the detailed information.

## Example

This example creates an outline with automatic styles, with the summary row above the detail rows, and with the summary column to the right of the detail columns.

```
Worksheets("Sheet1").Activate  
Selection.AutoOutline  
With ActiveSheet.Outline  
    .SummaryRow = xlAbove  
    .SummaryColumn = xlRight  
    .AutomaticStyles = True  
End With
```



# Superscript Property

**True** if the font is formatted as superscript; **False** by default. Read/write **Variant**.

## Example

This example makes the last character in cell A1 a superscript character.

```
n = Worksheets("Sheet1").Range("A1").Characters.Count  
Worksheets("Sheet1").Range("A1") _  
    .Characters(n, 1).Font.Superscript = True
```



# SurfaceGroup Property

Returns a [ChartGroup](#) object that represents the surface chart group of a 3-D chart. Read-only.

## Example

This example sets the 3-D surface group in Chart1 to use a different color for each data marker. The example should be run on a 3-D chart.

```
Charts("Chart1").SurfaceGroup.VaryByCategories = True
```



# Sync Property

Returns a **Sync** object that provides access to the methods and properties for documents that are part of a Document Workspace.

*expression*.**Sync**

*expression* Required. An expression that returns a [Workbook](#) object.

## Example

The following example displays the name of the last person to modify the active workbook if the active workbook is a shared document in a Document Workspace.

```
Dim eStatus As MsoSyncStatusType
Dim strLastUser As String

eStatus = ActiveDocument.Sync.Status

If eStatus = msoSyncStatusLatest Then
    strLastUser = ActiveWorkbook.Sync.WorkspaceLastChangedBy
    MsgBox "You have the most up-to-date copy." & _
        "This file was last modified by " & strLastUser
End If
```



# SyncScrollingSideBySide Property

**True** enables scrolling the contents of windows at the same time when documents are being compared side by side. **False** disables scrolling the windows at the same time.

*expression*.SyncScrollingSideBySide

*expression* Required. An expression that returns a **Windows** collection.



# Tab Property

Returns a [Tab](#) object for a chart or a worksheet.

*expression*.**Tab**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel determines if the worksheet's first tab color index is set to none and notifies the user.

```
Sub CheckTab()  
    ' Determine if color index of 1st tab is set to none.  
    If Worksheets(1).Tab.ColorIndex = xlColorIndexNone Then  
        MsgBox "The color index is set to none for the 1st " & _  
            "worksheet tab."  
    Else  
        MsgBox "The color index for the tab of the 1st worksheet " &  
            "is not set none."  
    End If  
End Sub
```



# TableRange1 Property

Returns a [Range](#) object that represents the range containing the entire PivotTable report, but doesn't include page fields. Read-only.

## Remarks

The [TableRange2](#) property includes page fields.

## Example

This example selects all of the PivotTable report except its page fields.

```
Worksheets("Sheet1").Activate  
Range("A3").PivotTable.TableRange1.Select
```



# TableRange2 Property

Returns a [Range](#) object that represents the range containing the entire PivotTable report, including page fields. Read-only.

## Remarks

The [TableRange1](#) property doesn't include page fields.

## Example

This example selects the entire PivotTable report, including its page fields.

```
Worksheets("Sheet1").Activate  
Range("A3").PivotTable.TableRange2.Select
```



# TableStyle Property

Returns or sets the style used in the body of the PivotTable report. The default value is a null string (no style is applied by default). Read/write **String**.

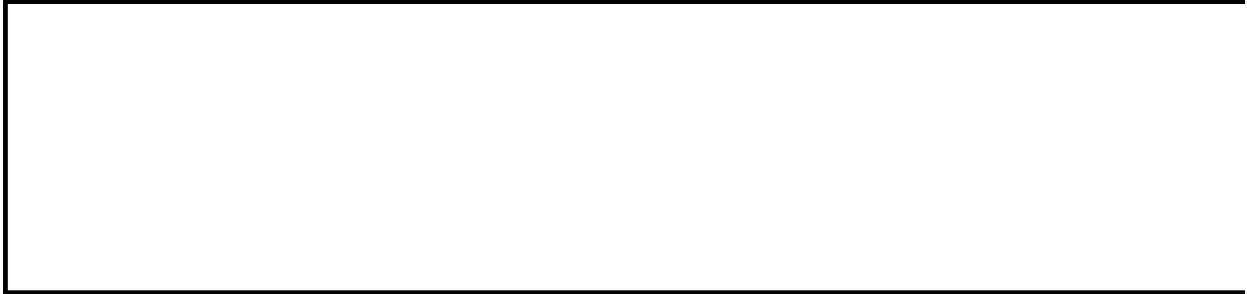
## **Remarks**

This style is used as the default style for the background area, and it's applied before any user formatting.

## Example

This example sets the body of the PivotTable report to the PurpleAndGold style.

```
Worksheets(1).PivotTables("Pivot1").TableStyle = "PurpleAndGold"
```



# TabRatio Property

Returns or sets the ratio of the width of the workbook's tab area to the width of the window's horizontal scroll bar (as a number between 0 (zero) and 1; the default value is 0.6). Read/write **Double**.

## Remarks

This property has no effect when [DisplayWorkbookTabs](#) is set to **False** (its value is retained, but it has no effect on the display).

## Example

This example makes the workbook tabs half the width of the horizontal scroll bar.

```
ActiveWindow.TabRatio = 0.5
```



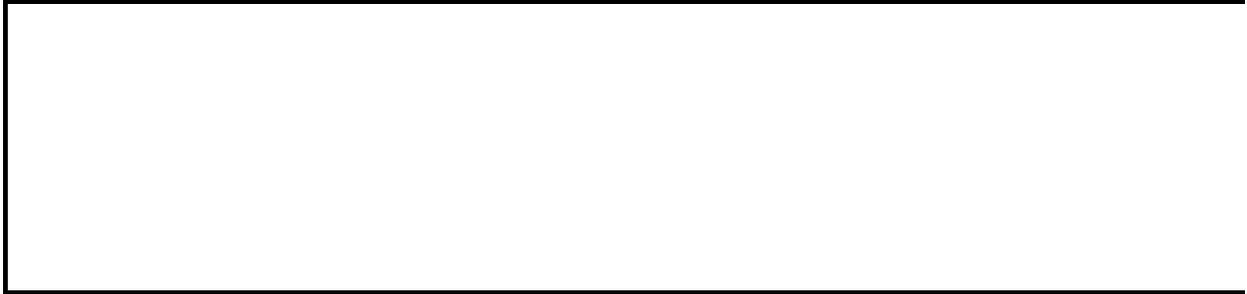
# Tag Property

Returns or sets a string saved with the PivotTable report. Read/write **String**.

## Example

This example sets the PivotTable report's **Tag** property.

```
Worksheets(1).PivotTables("Pivot1").Tag = "Product Sales by Region"
```



[Show All](#)

# TargetBrowser Property

Returns or sets an [MsoTargetBrowser](#) constant indicating the browser version. Read/write.

MsoTargetBrowser can be one of these MsoTargetBrowser constants.

**msoTargetBrowserIE4** Microsoft Internet Explorer 4.0 or later.

**msoTargetBrowserIE5** Microsoft Internet Explorer 5.0 or later.

**msoTargetBrowserIE6** Microsoft Internet Explorer 6.0 or later.

**msoTargetBrowserV3** Microsoft Internet Explorer 3.0, Netscape Navigator 3.0, or later.

**msoTargetBrowserV4** Microsoft Internet Explorer 4.0, Netscape Navigator 4.0, or later.

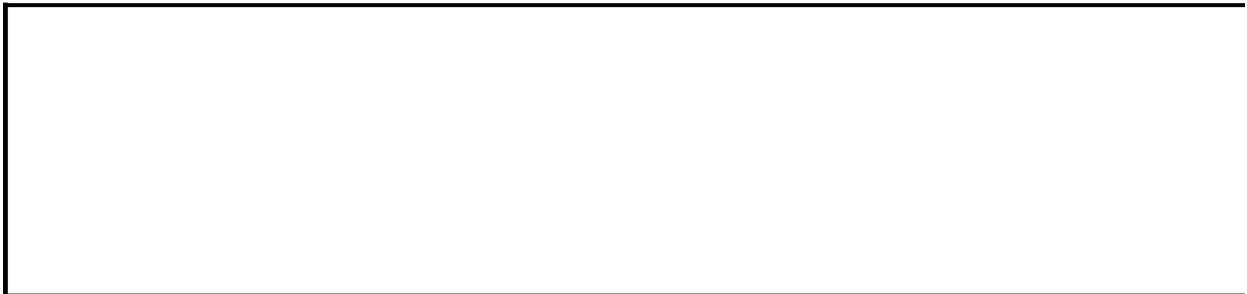
*expression*.**TargetBrowser**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel determines if the browser version for Web options is IE5 and notifies the user.

```
Sub CheckWebOptions()  
    Dim wkbOne As Workbook  
  
    Set wkbOne = Application.Workbooks(1)  
  
    ' Determine if IE5 is the target browser.  
    If wkbOne.WebOptions.TargetBrowser = msoTargetBrowserIE5 Then  
        MsgBox "The target browser is IE5 or later."  
    Else  
        MsgBox "The target browser is not IE5 or later."  
    End If  
  
End Sub
```



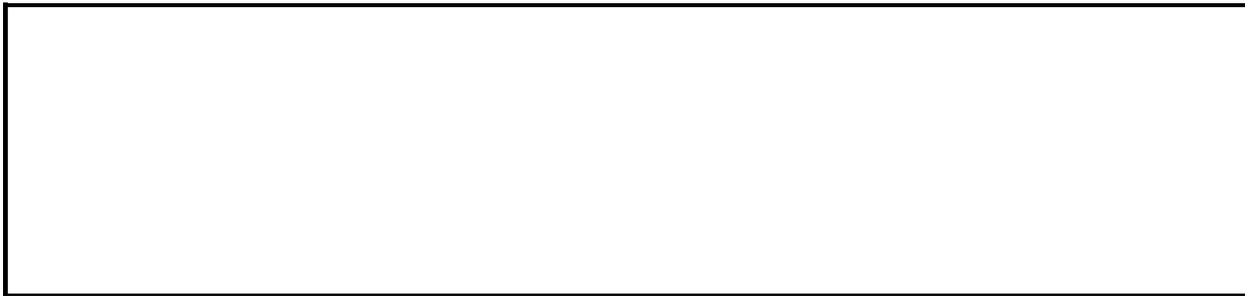
# TemplateRemoveExtData Property

**True** if external data references are removed when the workbook is saved as a template. Read/write **Boolean**.

## Example

This example saves the workbook as a template that contains no external data.

```
With ThisWorkbook
    .TemplateRemoveExtData = True
    .SaveAs "current", xlTemplate
    .TemplateRemoveExtData = False
End With
```



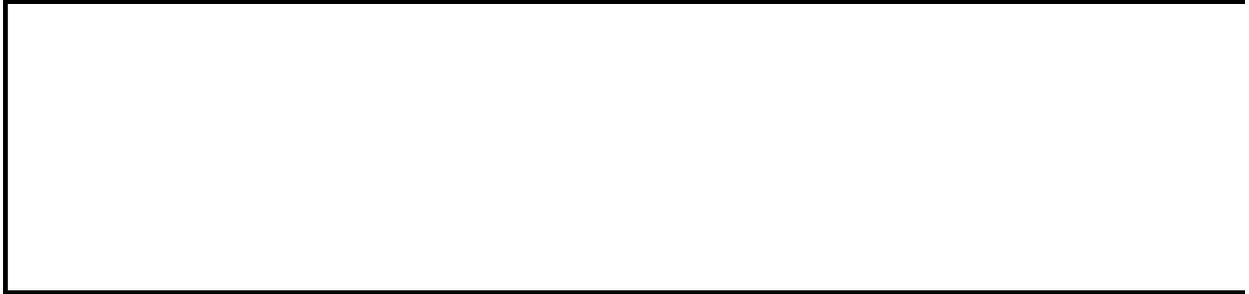
# TemplatesPath Property

Returns the local path where templates are stored. Read-only **String**.

## Example

This example returns the local path where templates are stored.

```
Msgbox Application.TemplatesPath
```



# Text Property

Returns or sets the text for the specified object. Read-only **String** for the **Range** object, read/write **String** for all other objects.

For information about using the **Text** worksheet function in Visual Basic, see [Using Worksheet Functions in Visual Basic](#).

## Remarks

For the [Phonetic](#) object, this property returns or sets its phonetic text. You cannot set this property to **Null**.

## Example

This example sets the text for the chart title of Chart1.

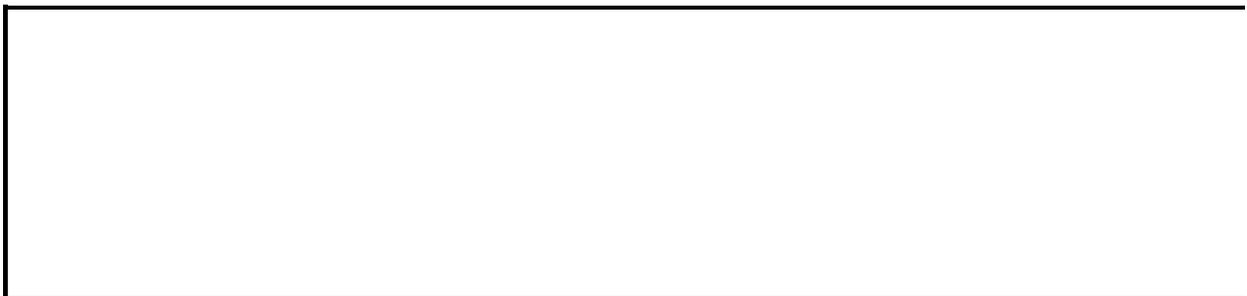
```
With Charts("Chart1")
    .HasTitle = True
    .ChartTitle.Text = "First Quarter Sales"
End With
```

This example sets the axis title text for the category axis in Chart1.

```
With Charts("Chart1").Axes(xlCategory)
    .HasTitle = True
    .AxisTitle.Text = "Month"
End With
```

This example illustrates the difference between the **Text** and **Value** properties of cells that contain formatted numbers.

```
Set c = Worksheets("Sheet1").Range("B14")
c.Value = 1198.3
c.NumberFormat = "$#,##0_);(,$#,##0)"
MsgBox c.Value
MsgBox c.Text
```



# TextboxText Property

Sets or returns a **String** that represents the text in a smart document textbox control. Read/write.

*expression*.**TextboxText**

*expression* Required. An expression that returns a **SmartTagAction** object.

## Remarks

For more information on smart documents, please see the Smart Document Software Development Kit on the [Microsoft Developer Network \(MSDN\)](#) Web site.

---

# TextDate Property

When set to **True** (default), Microsoft Excel identifies, with an **AutoCorrect Options** button, cells that contain a text date with a two-digit year. **False** disables error checking for cells containing a text date with a two-digit year. Read/write **Boolean**.

*expression*.**TextDate**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In the following example, the AutoCorrect Options button appears for cell A1, which contains a text date with a two-digit year.

```
Sub CheckTextDate()  
    ' Simulate an error by referencing a text date with a two-digit  
    Application.ErrorCheckingOptions.TextDate = True  
    Range("A1").Formula = "'April 23, 00"  
End Sub
```



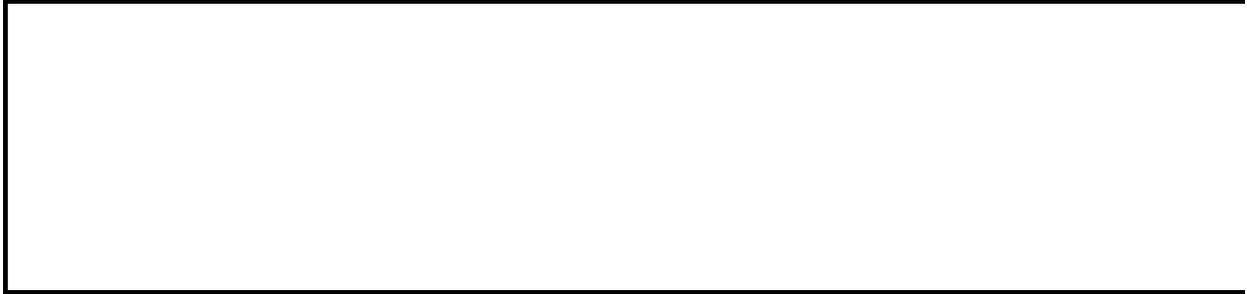
# TextEffect Property

Returns a [TextEffectFormat](#) object that contains text-effect formatting properties for the specified shape. Applies to **Shape** or **ShapeRange** objects that represent WordArt. Read-only.

## Example

This example sets the font style to bold for shape three on myDocument if the shape is WordArt.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(3)
    If .Type = msoTextEffect Then
        .TextEffect.FontBold = True
    End If
End With
```



# TextFileColumnDataTypes Property

Returns or sets an ordered array of constants that specify the data types applied to the corresponding columns in the text file that you're importing into a query table. The default constant for each column is **xlGeneral**. Read/write **Variant**.

You can use the **xlColumnDataType** constants listed in the following table to specify the column data types used or the actions taken during the data import.

<b>Constant</b>	<b>Description</b>
<b>xlGeneralFormat</b>	General
<b>xlTextFormat</b>	Text
<b>xlSkipColumn</b>	Skip column
<b>xlDMYFormat</b>	Day-Month-Year date format
<b>xlDYMFFormat</b>	Day-Year-Month date format
<b>xlEMDFFormat</b>	EMD date
<b>xlMDYFormat</b>	Month-Day-Year date format
<b>xlMYDFFormat</b>	Month-Year-Day date format
<b>xlYDMFormat</b>	Year-Day-Month date format
<b>xlYMDFormat</b>	Year-Month-Day date format

## Remarks

Use this property only when your query table is based on data from a text file (with the [QueryType](#) property set to **xlTextImport**).

If you specify more elements in the array than there are columns, those values are ignored.

You can use **xlEMDFormat** only if Taiwanese language support is installed and selected. The **xlEMDFormat** constant specifies that Taiwanese era dates are being used.

## Example

This example imports a fixed-width text file into a new query table on the first worksheet in the first workbook. The first column in the text file is five characters wide and is imported as text. The second column is four characters wide and is skipped. The remainder of the text file is imported into the third column and has the General format applied to it.

```
Set shFirstQtr = Workbooks(1).Worksheets(1)
Set qtQtrResults = shFirstQtr.QueryTables _
    .Add(Connection := "TEXT;C:\My Documents\19980331.txt", _
        Destination := shFirstQtr.Cells(1, 1))
With qtQtrResults
    .TextFileParseType = xlFixedWidth
    .TextFileFixedColumnWidths = Array(5, 4)
    .TextFileColumnDataTypes = _
        Array(xlTextFormat, xlSkipColumn, xlGeneralFormat)
    .Refresh
End With
```



# TextFileCommaDelimiter Property

**True** if the comma is the delimiter when you import a text file into a query table.  
**False** if you want to use some other character as the delimiter. The default value is **False**. Read/write **Boolean**.

## Remarks

Use this property only when your query table is based on data from a text file (with the [QueryType](#) property set to **xlTextImport**), and only if the value of the [TextFileParseType](#) property is **xlDelimited**.

## Example

This example sets the comma to be the delimiter in the query table on the first worksheet in the first workbook, and then it refreshes the query table.

```
Set shFirstQtr = Workbooks(1).Worksheets(1)
Set qtQtrResults = shFirstQtr.QueryTables _
    .Add(Connection := "TEXT;C:\My Documents\19980331.txt", _
        Destination := shFirstQtr.Cells(1, 1))
With qtQtrResults
    .TextFileParseType = xlDelimited
    .TextFileCommaDelimiter = True
    .Refresh
End With
```



# TextFileConsecutiveDelimiter Property

**True** if consecutive delimiters are treated as a single delimiter when you import a text file into a query table. The default value is **False**. Read/write **Boolean**.

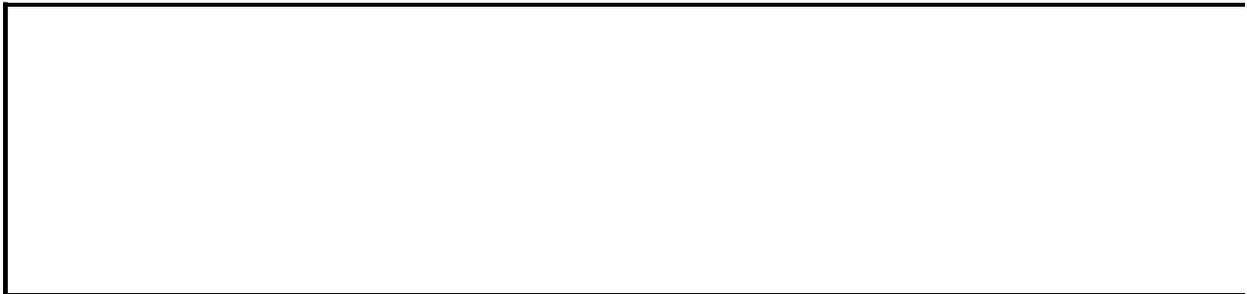
## Remarks

Use this property is only when your query table is based on data from a text file (with the [QueryType](#) property set to **xlTextImport**), and only if the value of the [TextFileParseType](#) property is **xlDelimited**.

## Example

This example sets the space character to be the delimiter in the query table on the first worksheet in the first workbook, and then it refreshes the query table. Consecutive spaces are treated as a single delimiter.

```
Set shFirstQtr = Workbooks(1).Worksheets(1)
Set qtQtrResults = shFirstQtr.QueryTables _
    .Add(Connection := "TEXT;C:\My Documents\19980331.txt", _
        Destination := shFirstQtr.Cells(1, 1))
With qtQtrResults
    .TextFileParseType = xlDelimited
    .TextFileSpaceDelimiter = True
    .TextFileConsecutiveDelimiter = True
    .Refresh
End With
```



# TextFileDecimalSeparator Property

Returns or sets the decimal separator character that Microsoft Excel uses when you import a text file into a query table. The default is the system decimal separator character. Read/write **String**.

## Remarks

Use this property only when your query table is based on data from a text file (with the [QueryType](#) property set to **xlTextImport**), when the file contains decimal and thousands separators that are different from those used on the computer, due to a different language setting being used.

The following table shows the results when you import text into Microsoft Excel using various separators. Numeric results are displayed in the rightmost column.

<b>System decimal separator</b>	<b>System thousands separator</b>	<b>TextFileDecimalSeparator value</b>	<b>TextFileThousandsSeparator value</b>
Period	Comma	Comma	Period
Period	Comma	Comma	Comma
Comma	Period	Comma	Period
Period	Comma	Period	Comma
Period	Comma	Period	Space

## Example

This example saves the original decimal separator and sets it to a comma for the first query table on Sheet1, in preparation for importing a French text file (for example) into the U.S. English version of Microsoft Excel.

```
strDecSep = Worksheets("Sheet1").QueryTables(1) _  
    .TextFileDecimalSeparator  
Worksheets("Sheet1").QueryTables(1) _  
    .TextFileDecimalSeparator = ","
```



# TextFileFixedColumnWidths Property

Returns or sets an array of integers that correspond to the widths of the columns (in characters) in the text file that you're importing into a query table. Valid widths are from 1 through 32767 characters. Read/write **Variant**.

## Remarks

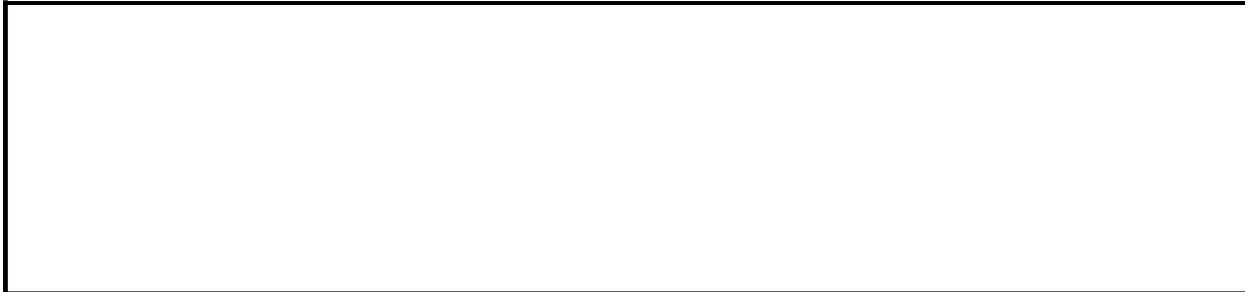
Use this property only when your query table is based on data from a text file (with the [QueryType](#) property set to **xlTextImport**), and only if the value of the [TextFileParseType](#) property is **xlFixedWidth**.

You must specify a valid, nonnegative column width. If you specify columns that exceed the width of the text file, those values are ignored. If the width of the text file is greater than the total width of columns you specify, the balance of the text file is imported into an additional column.

## Example

This example imports a fixed-width text file into a new query table on the first worksheet in the first workbook. The first column in the text file is five characters wide and is imported as text. The second column is four characters wide and is skipped. The remainder of the text file is imported into the third column and has the General format applied to it.

```
Set shFirstQtr = Workbooks(1).Worksheets(1)
Set qtQtrResults = shFirstQtr.QueryTables _
    .Add(Connection := "TEXT;C:\My Documents\19980331.txt", _
        Destination := shFirstQtr.Cells(1, 1))
With qtQtrResults
    .TextFileParseType = xlFixedWidth
    .TextFileFixedColumnWidths = Array(5, 4)
    .TextFileColumnDataTypes = _
        Array(xlTextFormat, xlSkipColumn, xlGeneralFormat)
    .Refresh
End With
```



# TextFileOtherDelimiter Property

Returns or sets the character used as the delimiter when you import a text file into a query table. The default value is **Null**. Read/write **String**.

## Remarks

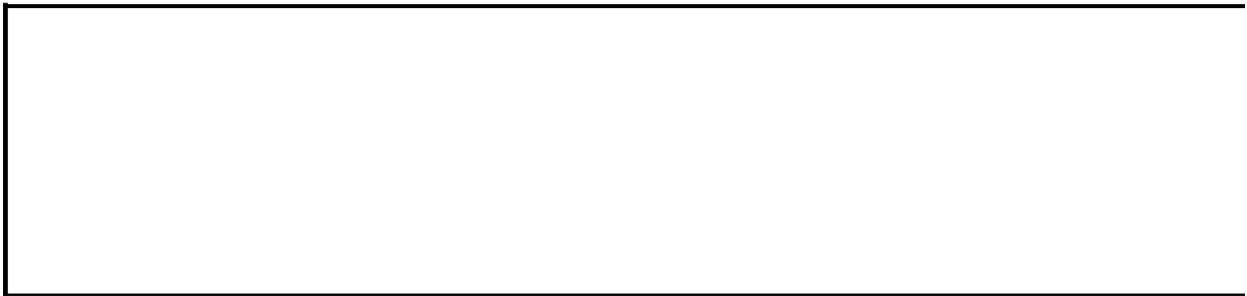
Use this property only when your query table is based on data from a text file (with the [QueryType](#) property set to **xlTextImport**), and only if the value of the [TextFileParseType](#) property is **xlDelimited**.

If you specify more than one character in the string, only the first character is used.

## Example

This example sets the pound character (#) to be the delimiter for the query table on the first worksheet in the first workbook, and then it refreshes the query table.

```
Set shFirstQtr = Workbooks(1).Worksheets(1)
Set qtQtrResults = shFirstQtr.QueryTables _
    .Add(Connection := "TEXT;C:\My Documents\19980331.txt", _
        Destination := shFirstQtr.Cells(1,1))
With qtQtrResults
    .TextFileParseType = xlDelimited
    .TextFileOtherDelimiter = "#"
    .Refresh
End With
```



[Show All](#)

# TextFileParseType Property

Returns or sets the column format for the data in the text file that you're importing into a query table. Read/write [XLTextParsingType](#).

XLTextParsingType can be one of these XLTextParsingType constants.

**xlFixedWidth**. Indicates that the data in the file is arranged in columns of fixed widths.

**xlDelimited** *default*. Indicates the file is delimited by delimiter characters

*expression*.**TextFileParseType**

*expression* Required. An expression that returns one of the objects in the Applies To list.

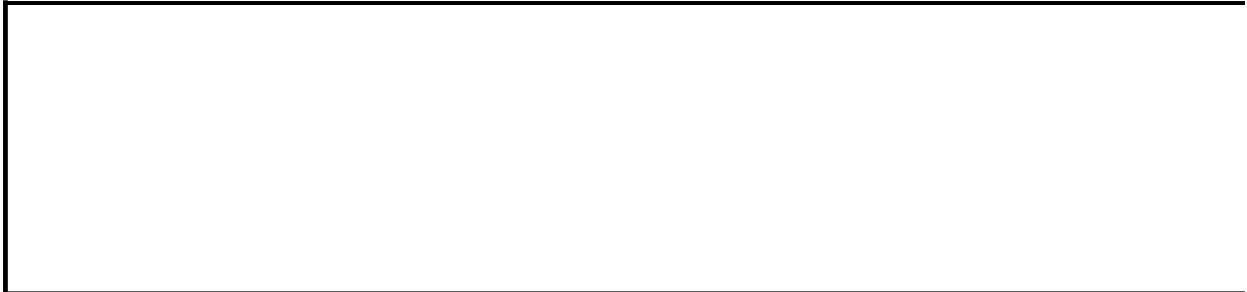
## Remarks

Use this property only when your query table is based on data from a text file (with the [QueryType](#) property set to **xlTextImport**).

## Example

This example imports a fixed-width text file into a new query table on the first worksheet in the first workbook. The first column in the text file is five characters wide and is imported as text. The second column is four characters wide and is skipped. The remainder of the text file is imported into the third column and has the General format applied to it.

```
Set shFirstQtr = Workbooks(1).Worksheets(1)
Set qtQtrResults = shFirstQtr.QueryTables _
    .Add(Connection := "TEXT;C:\My Documents\19980331.txt", _
        Destination := shFirstQtr.Cells(1, 1))
With qtQtrResults
    .TextFileParseType = xlFixedWidth
    .TextFileFixedColumnWidths = Array(5, 4)
    .TextFileColumnDataTypes = _
        Array(xlTextFormat, xlSkipColumn, xlGeneralFormat)
    .Refresh
End With
```



[Show All](#)

# TextFilePlatform Property

Returns or sets the origin of the text file you're importing into the query table. This property determines which code page is used during the data import. The default value is the current setting of the **File Origin** option in the Text File Import Wizard. Read/write [XIPlatform](#).

XIPlatform can be one of these XIPlatform constants.

**xlMacintosh**

**xlMSDOS**

**xlWindows**

*expression*.**TextFilePlatform**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Use this property only when your query table is based on data from a text file (with the [QueryType](#) property set to **xlTextImport**).

## Example

This example imports an MS-DOS text file into the query table on the first worksheet in the first workbook, and then it refreshes the query table.

```
Set shFirstQtr = Workbooks(1).Worksheets(1)
Set qtQtrResults = shFirstQtr.QueryTables _
    .Add(Connection := "TEXT;C:\My Documents\19980331.txt", _
        Destination := shFirstQtr.Cells(1,1))
With qtQtrResults
    .TextFilePlatform = xlMSDOS
    .TextFileParseType = xlDelimited
    .TextFileTabDelimiter = True
    .Refresh
End With
```



# TextFilePromptOnRefresh Property

**True** if you want to specify the name of the imported text file each time the query table is refreshed. The **Import Text File** dialog box allows you to specify the path and file name. The default value is **False**. Read/write **Boolean**.

## Remarks

Use this property only when your query table is based on data from a text file (with the [QueryType](#) property set to **xlTextImport**).

If the value of this property is **True**, the dialog box doesn't appear the first time a query table is refreshed.

The default value is **True** in the user interface.

## Example

This example prompts the user for the name of the text file whenever the query table on the first worksheet in the first workbook is refreshed.

```
Set shFirstQtr = Workbooks(1).Worksheets(1)
Set qtQtrResults = shFirstQtr.QueryTables _
    .Add(Connection := "TEXT;C:\My Documents\19980331.txt", _
        Destination := shFirstQtr.Cells(1,1))
With qtQtrResults
    .TextFileParseType = xlDelimited
    .TextFilePromptOnRefresh = True
    .TextFileTabDelimiter = True
    .Refresh
End With
```



# TextFileSemicolonDelimiter Property

**True** if the semicolon is the delimiter when you import a text file into a query table, and if the value of the [TextFileParseType](#) property is **xlDelimited**. The default value is **False**. Read/write **Boolean**.

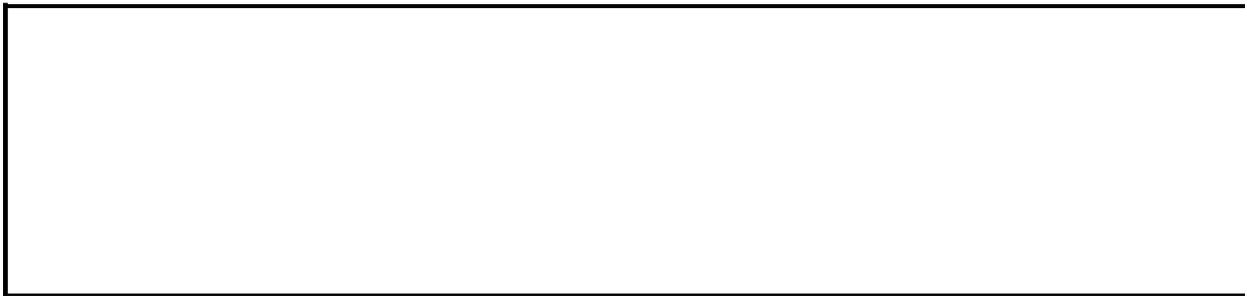
## Remarks

Use this property only when your query table is based on data from a text file (with the [QueryType](#) property set to **xlTextImport**).

## Example

This example sets the semicolon to be the delimiter in the query table on the first worksheet in the first workbook and then refreshes the query table.

```
Set shFirstQtr = Workbooks(1).Worksheets(1)
Set qtQtrResults = shFirstQtr.QueryTables _
    .Add(Connection := "TEXT;C:\My Documents\19980331.txt", _
        Destination := shFirstQtr.Cells(1,1))
With qtQtrResults
    .TextFileParseType = xlDelimited
    .TextFileSemicolonDelimiter = True
    .Refresh
End With
```



# TextFileSpaceDelimiter Property

**True** if the space character is the delimiter when you import a text file into a query table. The default value is **False**. Read/write **Boolean**.

## Remarks

Use this property only when your query table is based on data from a text file (with the [QueryType](#) property set to **xlTextImport**), and only if the value of the [TextFileParseType](#) property is **xlDelimited**.

## Example

This example sets the space character to be the delimiter in the query table on the first worksheet in the first workbook and then refreshes the query table.

```
Set shFirstQtr = Workbooks(1).Worksheets(1)
Set qtQtrResults = shFirstQtr.QueryTables _
    .Add(Connection := "TEXT;C:\My Documents\19980331.txt", _
        Destination := shFirstQtr.Cells(1,1))
With qtQtrResults
    .TextFileParseType = xlDelimited
    .TextFileSpaceDelimiter = True
    .Refresh
End With
```



# TextFileStartRow Property

Returns or sets the row number at which text parsing will begin when you import a text file into a query table. Valid values are integers from 1 through 32767. The default value is 1. Read/write **Long**.

## Remarks

Use this property only when your query table is based on data from a text file (with the [QueryType](#) property set to **xlTextImport**).

## Example

This example sets row 5 as the starting row for text parsing in the query table on the first worksheet in the first workbook, and then it refreshes the query table.

```
Set shFirstQtr = Workbooks(1).Worksheets(1)
Set qtQtrResults = shFirstQtr.QueryTables _
    .Add(Connection := "TEXT;C:\My Documents\19980331.txt", _
        Destination := shFirstQtr.Cells(1,1))
With qtQtrResults
    .TextFileParseType = xlDelimited
    .TextFileStartRow = 5
    .TextFileTabDelimiter = True
    .Refresh
End With
```



# TextFileTabDelimiter Property

**True** if the tab character is the delimiter when you import a text file into a query table. The default value is **False**. Read/write **Boolean**.

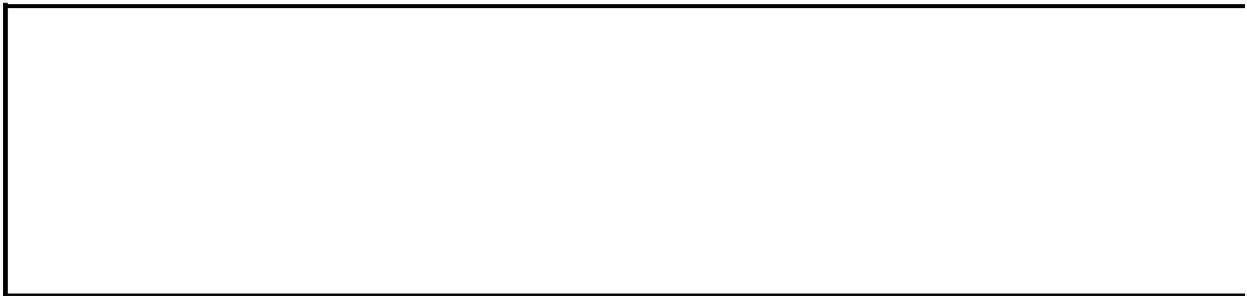
## Remarks

Use this property only when your query table is based on data from a text file (with the [QueryType](#) property set to **xlTextImport**), and only if the value of the [TextFileParseType](#) property is **xlDelimited**.

## Example

This example sets the tab character to be the delimiter in the query table on the first worksheet in the first workbook, and then it refreshes the query table.

```
Set shFirstQtr = Workbooks(1).Worksheets(1)
Set qtQtrResults = shFirstQtr.QueryTables _
    .Add(Connection := "TEXT;C:\My Documents\19980331.txt", _
        Destination := shFirstQtr.Cells(1,1))
With qtQtrResults
    .TextFileParseType = xlDelimited
    .TextFileTabDelimiter = True
    .Refresh
End With
```



[Show All](#)

# TextFileTextQualifier Property

Returns or sets the text qualifier when you import a text file into a query table. The text qualifier specifies that the enclosed data is in text format. Read/write [XLTextQualifier](#).

XLTextQualifier can be one of these XLTextQualifier constants.

**xlTextQualifierNone**

**xlTextQualifierDoubleQuote** *default*.

**xlTextQualifierSingleQuote**

*expression*.**TextFileTextQualifier**

*expression* Required. An expression that returns one of the objects in the Applies To list.

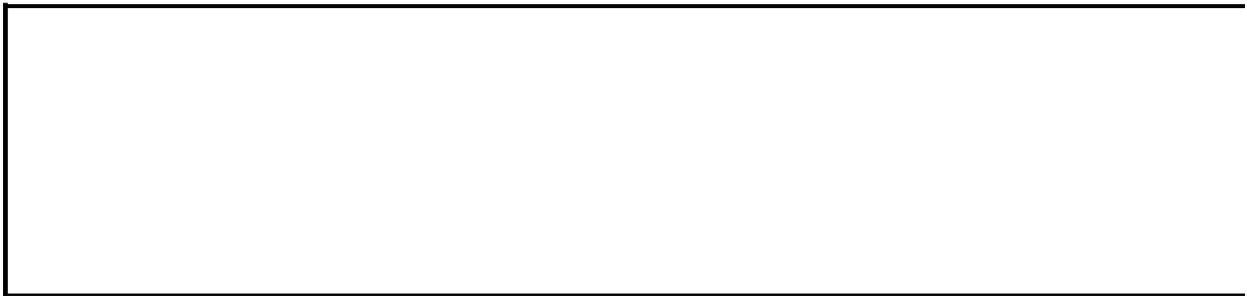
## Remarks

Use this property only when your query table is based on data from a text file (with the [QueryType](#) property set to **xlTextImport**).

## Example

This example sets the single quotation mark character as the text qualifier for the query table on the first worksheet in the first workbook.

```
Set shFirstQtr = Workbooks(1).Worksheets(1)
Set qtQtrResults = shFirstQtr.QueryTables _
    .Add(Connection := "TEXT;C:\My Documents\19980331.txt", _
        Destination := shFirstQtr.Cells(1,1))
With qtQtrResults
    .TextFileParseType = xlDelimited
    .TextFileTextQualifier = xlTextQualifierSingleQuote
    .Refresh
End With
```



# TextFileThousandsSeparator Property

Returns or sets the thousands separator character that Microsoft Excel uses when you import a text file into a query table. The default is the system thousands separator character. Read/write **String**.

## Remarks

Use this property only when your query table is based on data from a text file (with the [QueryType](#) property set to **xlTextImport**), especially when the file contains decimal and thousands separators that are different from those used on the computer, due to a different language setting being used.

The following table shows the results when you import text into Microsoft Excel using various separators. Numeric results are displayed in the rightmost column.

<b>System decimal separator</b>	<b>System thousands separator</b>	<b>TextFileDecimalSeparator value</b>	<b>TextFileThousandsSeparator value</b>
Period	Comma	Comma	Period
Period	Comma	Comma	Comma
Comma	Period	Comma	Period
Period	Comma	Period	Comma
Period	Comma	Period	Space

## Example

This example saves the original thousands separator and sets it to a period for the first query table on Sheet1, in preparation for importing a French text file (for example) into the U.S. English version of Microsoft Excel.

```
strDecSep = Worksheets("Sheet1").QueryTables(1) _  
    .TextFileThousandsSeparator  
Worksheets("Sheet1").QueryTables(1) _  
    .TextFileThousandsSeparator = "."
```



# TextFileTrailingMinusNumbers Property

**True** for Microsoft Excel to treat numbers imported as text that begin with a "-" symbol as a negative symbol. **False** for Excel to treat numbers imported as text that begin with a "-" symbol as text. Read/write **Boolean**.

*expression*. **TextFileTrailingMinusNumbers**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel determines the setting for cell A1, treating numbers imported as text that begin with a "-" symbol. This example assumes a [QueryTable](#) object exists on the active worksheet.

```
Sub CheckQueryTableSetting()  
    ' Determine setting for TextFileTrailingMinusNumbers  
    If Range("A1").QueryTable.TextFileTrailingMinusNumbers = True Th  
        MsgBox "Numbers imported as text that begin with a '-' symbo  
            "will be treated as a negative symbol."  
    Else  
        MsgBox "Numbers imported as text that begin with a '-' symbo  
            "will not be treated as a negative symbol."  
    End If  
End Sub
```



# TextFileVisualLayout Property

A **Long** value of either 1 or 2 that indicates the whether the visual layout of the text is left-to-right (1) or right-to-left (2) . Read/write **Long**.

*expression*.**TextFileVisualLayout**

*expression* Required. An expression that returns one of the objects in the Applies To list.



# TextFrame Property

Returns a [TextFrame](#) object that contains the alignment and anchoring properties for the specified shape. Read-only.

## Example

This example causes text in the text frame in shape one to be justified. If shape one doesn't have a text frame, this example fails.

```
Worksheets(1).Shapes(1).TextFrame _  
    .HorizontalAlignment = xlHAlignJustify
```



# TextShape Property

Returns a [Shape](#) object representing the shape of the text box associated with a diagram node.

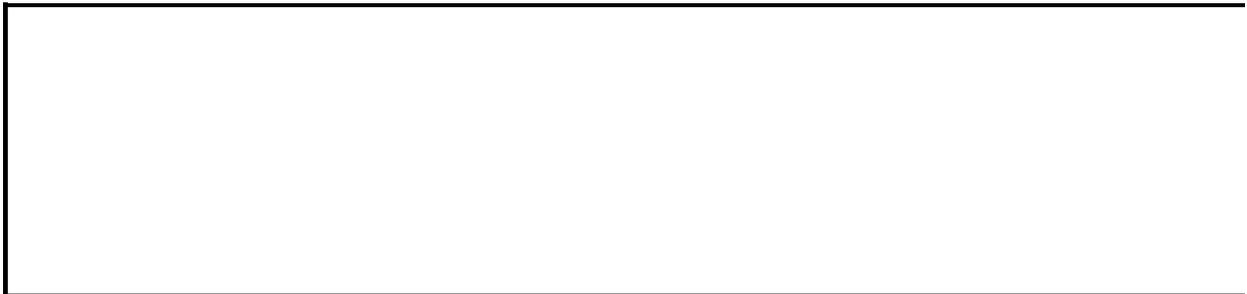
*expression*.**TextShape**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

The following example adds child nodes to a parent node, and displays text in the parent node indicating the number of child nodes created.

```
Sub CountChildNodes()  
  
    Dim nodRoot As DiagramNode  
    Dim shDiagram As Shape  
    Dim intCount As Integer  
    Dim shText As Shape  
  
    Set shDiagram = ActiveSheet.Shapes.AddDiagram _  
        (Type:=msoDiagramRadial, Left:=10, Top:=15, _  
         Width:=400, Height:=475)  
    Set nodRoot = shDiagram.DiagramNode.Children.AddNode  
  
    ' Add 3 child nodes to the root node.  
    For intCount = 1 To 3  
        nodRoot.Children.AddNode  
    Next  
  
    ' Change text in node.  
    For intCount = 1 To 4  
        Set shText = shDiagram.DiagramNode.Children.Item(1).TextShap  
        shText.TextFrame.Characters.Text = Str(intcount)  
    Next intCount  
  
End Sub
```



# TextToDisplay Property

Returns or sets the text to be displayed for the specified hyperlink. The default value is the address of the hyperlink. Read/write **String**.

## Example

This example sets the text to be displayed for the first hyperlink on the active worksheet.

```
ActiveSheet.Hyperlinks(1).TextToDisplay = _  
    "Company Home Page"
```



# TextureName Property

Returns the name of the custom texture file for the specified fill. Read-only **String**.

Use the **UserPicture** or **UserTextured** method to set the texture file for the fill.

## Example

This example sets the fill format for chart two to the same style used for chart one.

```
Set c1f = Charts(1).ChartArea.Fill
If c1f.Type = msoFillTextured Then
    With Charts(2).ChartArea.Fill
        .Visible = True
        If c1f.TextureType = msoTexturePreset Then
            .PresetTextured c1f.PresetTexture
        Else
            .UserTextured c1f.TextureName
        End If
    End With
End If
```



[Show All](#)

# TextureType Property

Returns the texture type for the specified fill. Read-only [MsoTextureType](#).

MsoTextureType can be one of these MsoTextureType constants.

**msoTexturePreset**

**msoTextureTypeMixed**

**msoTextureUserDefined**

*expression*.**TextureType**

*expression* Required. An expression that returns one of the objects in the Applies To list.

Use the **UserTextured** method to set the texture type for the fill.

## Example

This example sets the fill format for chart two to the same style used for chart one.

```
Set c1f = Charts(1).ChartArea.Fill
If c1f.Type = msoFillTextured Then
    With Charts(2).ChartArea.Fill
        .Visible = True
        If c1f.TextureType = msoTexturePreset Then
            .PresetTextured c1f.PresetTexture
        Else
            .UserTextured c1f.TextureName
        End If
    End With
End If
```



# ThisCell Property

Returns the cell in which the user-defined function is being called from as a [Range](#) object.

*expression*.**ThisCell**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Users should not access properties or methods on the **Range** object when inside the user-defined function. Users can cache the **Range** object for later use and perform additional actions when the recalculation is finished.

## Example

In this example, a function called "UseThisCell" contains the **ThisCell** property to notify the user of the cell address.

```
Function UseThisCell()
```

```
    MsgBox "The cell address is: " & _  
        Application.ThisCell.Address
```

```
End Function
```



# ThisWorkbook Property

Returns a [Workbook](#) object that represents the workbook where the current macro code is running. Read-only.

## Remarks

Use this property to refer to the workbook that contains your macro code.

**ThisWorkbook** is the only way to refer to an add-in workbook from inside the add-in itself. The **ActiveWorkbook** property doesn't return the add-in workbook; it returns the workbook that's *calling* the add-in. The **Workbooks** property may fail, as the workbook name probably changed when you created the add-in. **ThisWorkbook** always returns the workbook in which the code is running.

For example, use code such as the following to activate a dialog sheet stored in your add-in workbook.

```
ThisWorkbook.DialogSheets(1).Show
```

This property can be used only from inside Microsoft Excel. You cannot use it to access a workbook from any other application.

## Example

This example closes the workbook that contains the example code. Changes to the workbook, if any, aren't saved.

```
ThisWorkbook.Close SaveChanges:=False
```



# ThousandsSeparator Property

Sets or returns the character used for the thousands separator as a **String**.  
Read/write.

*expression*.**ThousandsSeparator**

*expression* Required. An expression that returns an **Application** object.

## Example

This example places "1,234,567.89" in cell A1 then changes the system separators to dashes for the decimals and thousands separators.

```
Sub ChangeSystemSeparators()  
  
    Range("A1").Formula = "1,234,567.89"  
    MsgBox "The system separators will now change."  
  
    ' Define separators and apply.  
    Application.DecimalSeparator = "-"  
    Application.ThousandsSeparator = "-"  
    Application.UseSystemSeparators = False  
  
End Sub
```



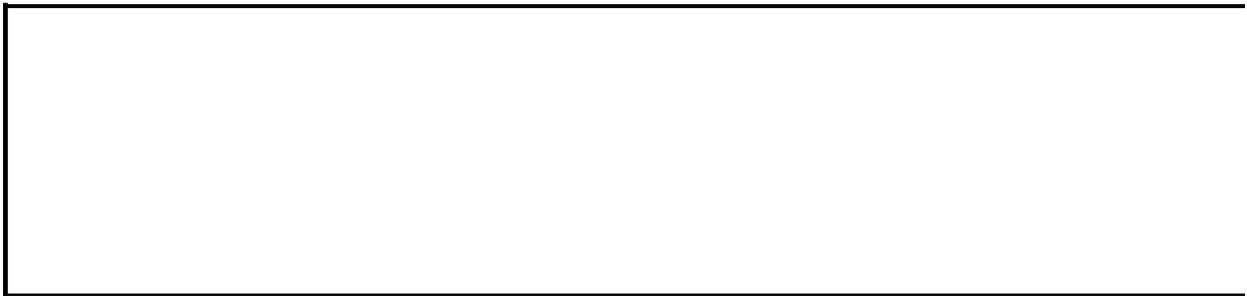
# ThreeD Property

Returns a [ThreeDFormat](#) object that contains 3-D – effect formatting properties for the specified shape. Read-only.

## Example

This example sets the depth, extrusion color, extrusion direction, and lighting direction for the 3-D effects applied to shape one on myDocument.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(1).ThreeD
    .Visible = True
    .Depth = 50
    .ExtrusionColor.RGB = RGB(255, 100, 255)
    ' RGB value for purple
    .SetExtrusionDirection msoExtrusionTop
    .PresetLightingDirection = msoLightingLeft
End With
```



# ThrottleInterval Property

Returns or sets a **Long** indicating the time interval between updates. Read/write.

*expression*.**ThrottleInterval**

*expression* Required. An expression that returns an **RTD** object.

## Remarks

The default value is 2000 milliseconds. If this value is changed, the new value will persist when Microsoft Excel is restarted.

--

[Show All](#)

# TickLabelPosition Property

Describes the position of tick-mark labels on the specified axis. Read/write [XlTickLabelPosition](#).

XlTickLabelPosition can be one of these XlTickLabelPosition constants.

**xlTickLabelPositionLow**

**xlTickLabelPositionNone**

**xlTickLabelPositionHigh**

**xlTickLabelPositionNextToAxis**

*expression*.**TickLabelPosition**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example sets tick-mark labels on the category axis in Chart1 to the high position (above the chart).

```
Charts("Chart1").Axes(xlCategory) _  
    .TickLabelPosition = xlTickLabelPositionHigh
```



# TickLabels Property

Returns a [TickLabels](#) object that represents the tick-mark labels for the specified axis. Read-only.

## Example

This example sets the color of the tick-mark label font for the value axis in Chart1.

```
Charts("Chart1").Axes(x1Value).TickLabels.Font.ColorIndex = 3
```



# TickLabelSpacing Property

Returns or sets the number of categories or series between tick-mark labels. Applies only to category and series axes. Read/write **Long**.

## Remarks

Tick-mark label spacing on the value axis is always calculated by Microsoft Excel.

## Example

This example sets the number of categories between tick-mark labels on the category axis in Chart1.

```
Charts("Chart1").Axes(xlCategory).TickLabelSpacing = 10
```



# TickMarkSpacing Property

Returns or sets the number of categories or series between tick marks. Applies only to category and series axes. Read/write **Long**.

## Remarks

Use the [MajorUnit](#) and [MinorUnit](#) properties to set tick-mark spacing on the value axis.

## Example

This example sets the number of categories between tick marks on the category axis in Chart1.

```
Charts("Chart1").Axes(xlCategory).TickMarkSpacing = 10
```



# Time Property

Sets or returns the time interval for the [AutoRecover](#) object. Permissible values are integers from 1 to 120 minutes. The default value is 10 minutes. Read/write **Long**.

*expression*.**Time**

*expression* Required. An expression that returns an [AutoRecover](#) object.

## **Remarks**

Entering a decimal value will round to the nearest whole number. For example, entering a value of 5.5 is the equivalent of 6.

If time values outside the valid range are entered, Microsoft Excel will revert to the previous time value used.

## Example

The following example sets the `AutoRecover` time interval to 5 minutes and notifies the user.

```
Sub SetTimeValue()  
    Application.AutoRecover.Time = 5  
    MsgBox "The AutoRecover time interval is set at " & _  
        Application.AutoRecover.Time & " minutes."  
End Sub
```



# TintAndShade Property

Returns or sets a **Single** that lightens or darkens the color of a specified shape.  
Read/write.

*expression*.**TintAndShade**

*expression* Required. An expression that returns one of the objects in the Applies To list.

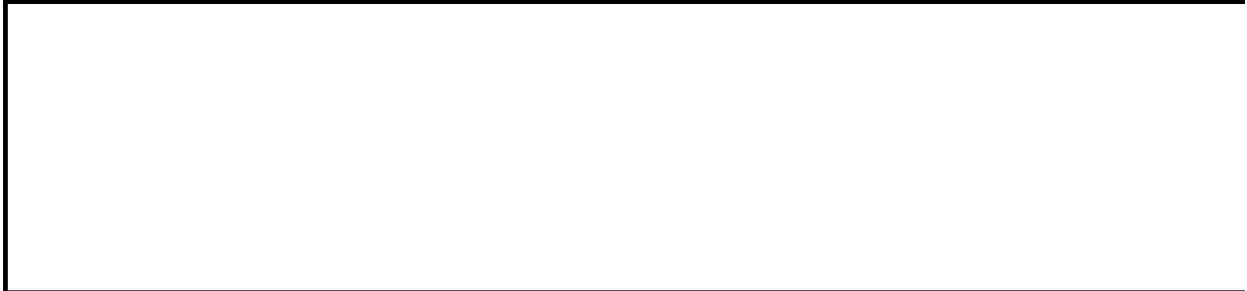
## Remarks

You can enter a number from -1 (darkest) to 1 (lightest) for the **TintAndShade** property, 0 (zero) being neutral.

## Example

This example creates a new shape in the active document, sets the fill color, and lightens the color shade.

```
Sub PrinterPlate()  
    Dim s As Shape  
  
    Set s = ActiveSheet.Shapes.AddShape( _  
        Type:=msoShapeHeart, Left:=150, _  
        Top:=150, Width:=250, Height:=250)  
  
    With s.Fill.ForeColor  
        .SchemeColor = 12  
        .TintAndShade = 0.8  
    End With  
  
End Sub
```



# Title Property

Returns or sets the title of the Web page when the document is saved as a Web page. Read/write **String**.

## **Remarks**

The title is usually displayed in the window title bar when the document is viewed in the Web browser.

## Example

This example sets the Web page title to "Sales Forecast" when the first item in the first workbook is saved as a Web page.

```
Workbooks(1).PublishObjects(1).Title = "Sales Forecast"
```



[Show All](#)

# Top Property

[Top property as it applies to the \*\*Application\*\* object.](#)

The distance from the top edge of the screen to the top edge of the main Microsoft Excel window. If the application window is minimized, this property controls the position of the window icon (anywhere on the screen). Read/write **Double**.

*expression*.**Top**

*expression* Required. An expression that returns an **Application** object.

[Top property as it applies to the \*\*Window\*\* object.](#)

The distance from the top edge of the window to the top edge of the usable area (below the menus, any toolbars docked at the top, and the formula bar). You cannot set this property for a maximized window. Use the **WindowState** property to return or set the state of the window. Read/write **Double**.

*expression*.**Top**

*expression* Required. An expression that returns a **Window** object.

[Top property as it applies to the \*\*AxisTitle\*\*, \*\*ChartArea\*\*, \*\*ChartObject\*\*, \*\*ChartObjects\*\*, \*\*ChartTitle\*\*, \*\*DataLabel\*\*, \*\*DisplayUnitLabel\*\*, \*\*Legend\*\*, \*\*OLEObject\*\*, \*\*OLEObjects\*\*, \*\*PlotArea\*\*, and \*\*Window\*\* objects.](#)

The distance from the top edge of the object to the top of row 1 (on a worksheet) or the top of the chart area (on a chart). Read/write **Double**.

*expression*.**Top**

*expression* Required. An expression that returns one of the above objects.

[Top property as it applies to the \*\*Axis\*\*, \*\*LegendEntry\*\*, and \*\*LegendKey\*\*](#)

[objects.](#)

The distance from the top edge of the object to the top of row 1 (on a worksheet) or the top of the chart area (on a chart). Read-only **Double**.

*expression*.**Top**

*expression* Required. An expression that returns one of the above objects.



[Top property as it applies to the \*\*Shape\*\* and \*\*ShapeRange\*\* objects.](#)

The distance from the top edge of the topmost shape in the shape range to the top edge of the worksheet. Read/write **Single**.

*expression*.**Top**

*expression* Required. An expression that returns one of the above objects.



[Top property as it applies to the \*\*Range\*\* object.](#)

The distance from the top edge of row 1 to the top edge of the range. If the range is discontinuous, the first area is used. If the range is more than one row high, the top (lowest numbered) row in the range is used. Read-only **Variant**.

*expression*.**Top**

*expression* Required. An expression that returns a **Range** object.

## Example

This example arranges windows one and two horizontally; in other words, each window occupies half the available vertical space and all the available horizontal space in the application window's client area. For this example to work, there must be only two worksheet windows open.

```
Windows.Arrange xlArrangeTiled
ah = Windows(1).Height           ' available height
aw = Windows(1).Width + Windows(2).Width ' available width
With Windows(1)
    .Width = aw
    .Height = ah / 2
    .Left = 0
End With
With Windows(2)
    .Width = aw
    .Height = ah / 2
    .Top = ah / 2
    .Left = 0
End With
```



# TopLeftCell Property

Returns a [Range](#) object that represents the cell that lies under the upper-left corner of the specified object. Read-only.

## Example

This example displays the address of the cell beneath the upper-left corner of embedded chart one on Sheet1.

```
MsgBox "The top left corner is over cell " & _  
    Worksheets("Sheet1").ChartObjects(1).TopLeftCell.Address
```



[Show All](#)

# TopMargin Property

Returns or sets the size of the top margin, in [points](#). Read/write **Double**.

## Remarks

Margins are set or returned in points. Use the **InchesToPoints** method or the **CentimetersToPoints** method to convert measurements from inches or centimeters.

## Example

These two examples set the top margin of Sheet1 to 0.5 inch (36 points).

```
Worksheets("Sheet1").PageSetup.TopMargin = _  
    Application.InchesToPoints(0.5)
```

```
Worksheets("Sheet1").PageSetup.TopMargin = 36
```

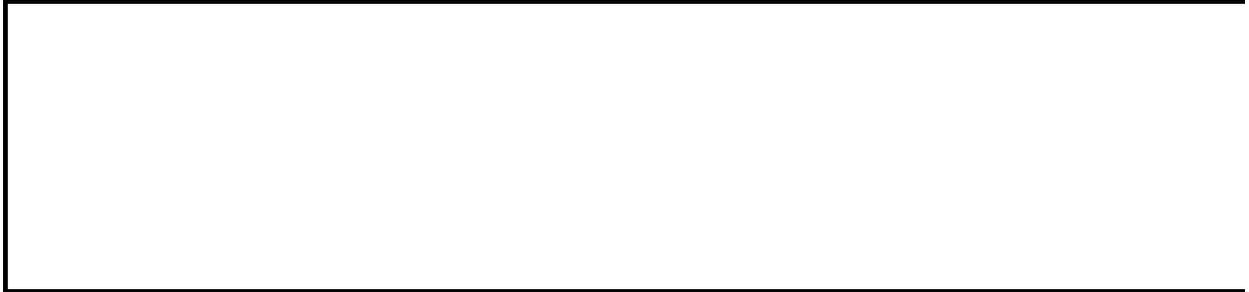
This example displays the current top-margin setting.

```
marginInches = ActiveSheet.PageSetup.TopMargin / _  
    Application.InchesToPoints(1)  
MsgBox "The current top margin is " & marginInches & " inches"
```



# ToRecipients Property

You have requested Help for a Visual Basic keyword used only on the Macintosh. For information about this keyword, consult the language reference Help included with Microsoft Office Macintosh Edition.



[Show All](#)

# TotalLevels Property

Returns the total number of fields in the current field group. If the field isn't grouped, or if the data source is [OLAP](#)-based, **TotalLevels** returns the value 1. Read-only **Long**.

## Remarks

All fields in a set of grouped fields have the same **TotalLevels** value.

## Example

This example displays the total number of fields in the group that contains the active cell.

```
Worksheets("Sheet1").Activate  
MsgBox "This group has " & _  
    ActiveCell.PivotField.TotalLevels & " levels."
```



# TotalsAnnotation Property

**True** if an asterisk (\*) is displayed next to each subtotal and grand total value in the specified PivotTable report if the report is based on an OLAP data source. The default value is **True**. Read/write **Boolean**.

## Remarks

When this property is set to **True**, the asterisk indicates that hidden items are included in the total. The asterisk appears regardless of whether any items in the report have been hidden.

For non-OLAP data sources, the value of this property is always **False**.

## Example

This example turns off the asterisks in the first PivotTable report on the active worksheet.

```
ActiveSheet.PivotTables(1).TotalsAnnotation = False
```



[Show All](#)

# TotalsCalculation Property

Determines the type of calculation in the Totals row of the list column based on the value of the **XITotalsCalculation** enumeration. Read/write.

XITotalsCalculation can be one of these XITotalsCalculation constants.

**xITotalsCalculationNone**

**xITotalsCalculationSum**

**xITotalsCalculationAverage**

**xITotalsCalculationCount**

**xITotalsCalculationCountNums**

**xITotalsCalculationMin**

**xITotalsCalculationStdDev**

**xITotalsCalculationVar**

**xITotalsCalculationMax**

*expression*.**TotalsCalculation**

*expression* Required. An expression that returns one of the items in the Applies To list.

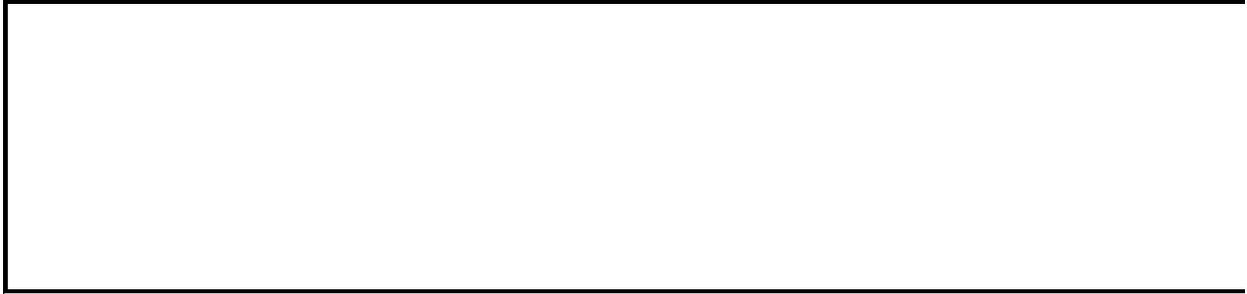
## Remarks

The Totals row doesn't need to be showing in order to set this property. There is no fixed "default" value for this property. Excel may change the state of this property, as other columns are added or deleted.

## Example

This example sets the calculation type of the Totals row in the active worksheet to sum.

```
ActiveSheet.ListColumns(1).TotalsCalculation=xlTotalsCalculationSum
```



# TotalsRowRange Property

Returns a **Range** representing the Total row, if any, from a specified **ListObject** object. Read-only.

*expression*.**TotalsRowRange**

*expression* Required. An expression that returns **ListObject** object.

## Example

The following sample code returns the address of the Total row in the default list in Sheet1 of the active workbook. The code displays the Total row if it is not displayed already.

```
Sub DisplayTotalsRowAddress()  
    Dim wrksht As Worksheet  
    Dim objListObj As ListObject  
  
    Set wrksht = ActiveWorkbook.Worksheets("Sheet2")  
    Set objListObj = wrksht.ListObjects(1)  
    objListObj.ShowTotals = True  
    MsgBox objListObj.TotalsRowRange.Address  
End Sub
```



# Tracking Property

Returns or sets the ratio of the horizontal space allotted to each character in the specified WordArt to the width of the character. Can be a value from 0 (zero) through 5. (Large values for this property specify ample space between characters; values less than 1 can produce character overlap.) Read/write **Single**.

The following table gives the values of the **Tracking** property that correspond to the settings available in the user interface.

<b>User interface setting</b>	<b>Equivalent Tracking property value</b>
Very Tight	0.8
Tight	0.9
Normal	1.0
Loose	1.2
Very Loose	1.5

## Example

This example adds WordArt that contains the text "Test" to myDocument and specifies that the characters be very tightly spaced.

```
Set myDocument = Worksheets(1)
Set newWordArt = myDocument.Shapes.AddTextEffect( _
    PresetTextEffect:=msoTextEffect1, Text:="Test", _
    FontName:="Arial Black", FontSize:=36, _
    FontBold:=False, FontItalic:=False, Left:=100, _
    Top:=100)
newWordArt.TextEffect.Tracking =0.8
```



# TrackStatus Property

**True** if status tracking is enabled for the routing slip. Read/write **Boolean**.

## Remarks

You cannot set this property if routing is in progress

## Example

This example sends Book1.xls to three recipients, with status tracking enabled.

```
Workbooks("BOOK1.XLS").HasRoutingSlip = True
With Workbooks("BOOK1.XLS").RoutingSlip
    .Delivery = xlOneAfterAnother
    .Recipients = Array("Adam Bendel", _
        "Jean Selva", "Bernard Gabor")
    .Subject = "Here is BOOK1.XLS"
    .Message = "Here is the workbook. What do you think?"
    .ReturnWhenDone = True
    .TrackStatus = True
End With
Workbooks("BOOK1.XLS").Route
```



# TransitionExpEval Property

**True** if Microsoft Excel uses Lotus 1-2-3 expression evaluation rules for the worksheet. Read/write **Boolean**.

## Example

This example causes Microsoft Excel to use Lotus 1-2-3 expression evaluation rules for Sheet1.

```
Worksheets("Sheet1").TransitionExpEval = True
```



# TransitionFormEntry Property

**True** if Microsoft Excel uses Lotus 1-2-3 formula entry rules for the worksheet.  
Read/write **Boolean**.

## Example

This example causes Microsoft Excel to use Lotus 1-2-3 formula entry rules for Sheet1.

```
Worksheets("Sheet1").TransitionFormEntry = True
```



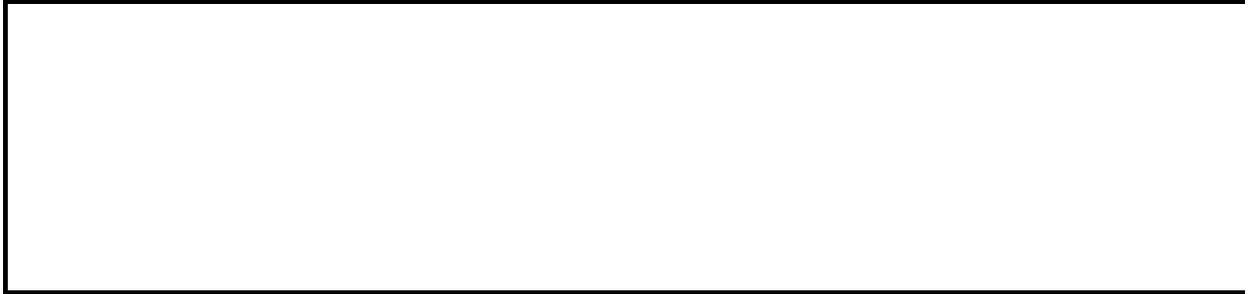
# TransitionMenuKey Property

Returns or sets the Microsoft Excel menu or help key, which is usually "/".  
Read/write **String**.

## Example

This example sets the transition menu key to "/" (which is the default).

```
Application.TransitionMenuKey = "/"
```



# TransitionMenuKeyAction Property

Returns or sets the action taken when the Microsoft Excel menu key is pressed. Can be either **xlExcelMenus** or **xlLotusHelp**. Read/write **Long**.

## Example

This example sets the Microsoft Excel menu key to run Lotus 1-2-3 Help when it is pressed.

```
Application.TransitionMenuKeyAction = xlLotusHelp
```



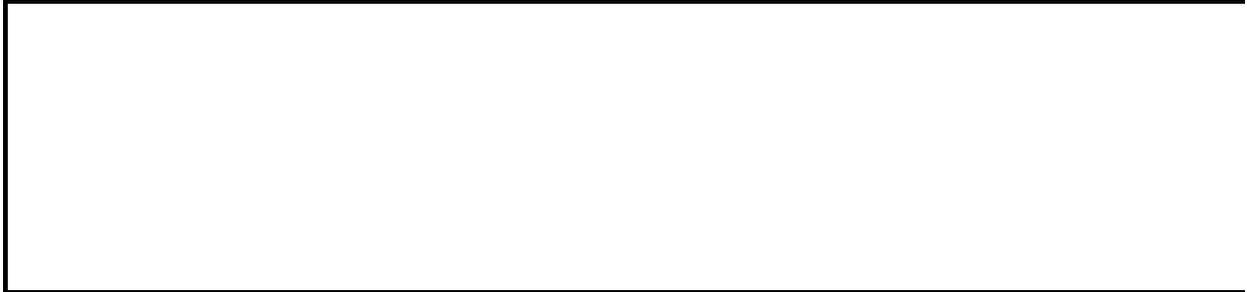
# TransitionNavigKeys Property

**True** if transition navigation keys are active. Read/write **Boolean**.

## Example

This example displays the current state of the **Transition navigation keys** option.

```
If Application.TransitionNavigKeys Then
    keyState = "On"
Else
    keyState = "Off"
End If
MsgBox "The Transition Navigation Keys option is " & keyState
```



# Transparency Property

Returns or sets the degree of transparency of the specified fill as a value from 0.0 (opaque) through 1.0 (clear). Read/write **Double**.

## Remarks

The value of this property affects the appearance of solid-colored fills and lines only; it has no effect on the appearance of patterned lines or patterned, gradient, picture, or textured fills.

## Example

This example sets the shadow of shape three on worksheet one to semitransparent red. If the shape doesn't already have a shadow, this example adds one to it.

```
With Worksheets(1).Shapes(3).Shadow  
    .Visible = True  
    .ForeColor.RGB = RGB(255, 0, 0)  
    .Transparency = 0.5  
End With
```



# TransparencyColor Property

Returns or sets the transparent color for the specified picture as a red-green-blue (RGB) value. For this property to take effect, the [TransparentBackground](#) property must be set to **True**. Applies to bitmaps only. Read/write **Long**.

*expression*.**TransparencyColor**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

If you want to be able to see through the transparent parts of the picture all the way to the objects behind the picture, you must set the **Visible** property of the picture's **FillFormat** object to **False**. If your picture has a transparent color and the **Visible** property of the picture's **FillFormat** object is set to **True**, the picture's fill will be visible through the transparent color, but objects behind the picture will be obscured.

## Example

This example sets the color that has the RGB value returned by the function RGB(0, 0, 255) as the transparent color for shape one on myDocument. For the example to work, shape one must be a bitmap.

```
blueScreen = RGB(0, 0, 255)
Set myDocument = Worksheets(1)
With myDocument.Shapes(1)
    With .PictureFormat
        .TransparentBackground = True
        .TransparencyColor = blueScreen
    End With
    .Fill.Visible = False
End With
```



[Show All](#)

# TransparentBackground Property

Use the [TransparencyColor](#) property to set the transparent color. Applies to bitmaps only. Read/write [MsoTriState](#).

MsoTriState can be one of these MsoTriState constants.

**msoCTrue**

**msoFalse**

**msoTriStateMixed**

**msoTriStateToggle**

**msoTrue** The parts of the picture that are the color defined as the transparent color appear transparent.

## Remarks

If you want to be able to see through the transparent parts of the picture all the way to the objects behind the picture, you must set the **Visible** property of the picture's **FillFormat** object to **False**. If your picture has a transparent color and the **Visible** property of the picture's **FillFormat** object is set to **True**, the picture's fill will be visible through the transparent color, but objects behind the picture will be obscured.

## Example

This example sets the color that has the RGB value returned by the function RGB(0, 24, 240) as the transparent color for shape one on myDocument. For the example to work, shape one must be a bitmap.

```
blueScreen = RGB(0, 0, 255)
Set myDocument = Worksheets(1)
With myDocument.Shapes(1)
    With .PictureFormat
        .TransparentBackground = True
        .TransparencyColor = blueScreen
    End With
    .Fill.Visible = False
End With
```



[Show All](#)

# TreeviewControl Property

Returns the [TreeviewControl](#) object of the [CubeField](#) object, representing the [cube](#) manipulation control of an [OLAP](#)-based PivotTable report. Read-only.

## **Remarks**

This property is available only when the control is visible.

## Example

This example sets the first cube field control to “drilled” for the states of California and Maryland in the second PivotTable report on the active worksheet.

```
ActiveSheet.PivotTables("PivotTable2") _  
    .CubeFields(1).TreeviewControl.Drilled = _  
        Array(Array("", ""), _  
              Array("[state].[states].[CA]", _  
                    "[state].[states].[MD]"))
```



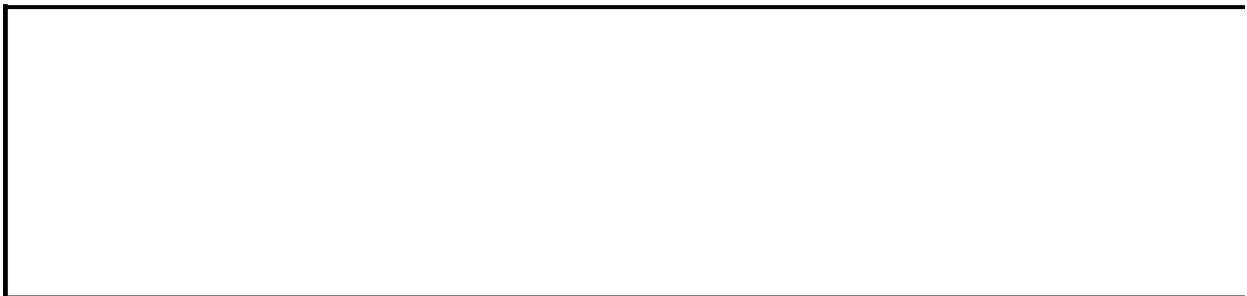
# TwoInitialCapitals Property

**True** if words that begin with two capital letters are corrected automatically.  
Read/write **Boolean**.

## Example

This example sets Microsoft Excel to correct words that begin with two capital letters.

```
With Application.AutoCorrect  
    .TwoInitialCapitals = True  
    .ReplaceText = True  
End With
```



[Show All](#)

# Type Property

 [Type property as it applies to the \*\*Axis\*\* object.](#)

Returns the Axis type. Read-only. [XlAxisType](#).

XlAxisType can be one of these XlAxisType constants.

**xlCategory**

**xlSeriesAxis**

**xlValue**

*expression.Type*

*expression* Required. An expression that returns an **Axis** object.

 [Type property as it applies to the \*\*CalculatedMember\*\* object.](#)

Returns the calculated member type. Read-only [XlCalculatedMemberType](#).

XlCalculatedMemberType can be one of these XlCalculatedMemberType constants.

**xlCalculatedMember**

**xlCalculatedSet**

*expression.Type*

*expression* Required. An expression that returns a **CalculatedMember** object.

 [Type property as it applies to the \*\*CalloutFormat\*\* object.](#)

Returns or sets the callout format type. Read/write [MsoCalloutType](#).

MsoCalloutType can be one of these MsoCalloutType constants.

**msoCalloutFour**

**msoCalloutMixed**

**msoCalloutOne**  
**msoCalloutThree**  
**msoCalloutTwo**

*expression.Type*

*expression* Required. An expression that returns a **CalloutFormat** object.

[Type property as it applies to the \*\*Chart\*\*, \*\*ChartGroup\*\*, and \*\*Series\*\* objects.](#)

Returns or sets the chart or series type. Read/write **Long**.

*expression.Type*

*expression* Required. An expression that returns one of the above objects.

[Type property as it applies to the \*\*ChartColorFormat\*\*, \*\*FormatCondition\*\*, \*\*Hyperlink\*\*, and \*\*Validation\*\* objects.](#)

Returns or sets the object type. Read-only **Long**.

*expression.Type*

*expression* Required. An expression that returns one of the above objects.

[Type property as it applies to the \*\*ChartFillFormat\*\* and \*\*FillFormat\*\* objects.](#)

Returns or sets the the fill type. Read-only **MsoFillType**.

MsoFillType can be one of these MsoFillType constants.

**msoFillBackground** This constant is not used in Microsoft Excel.

**msoFillGradient**

**msoFillMixed**

**msoFillPatterned**

**msoFillPicture**

**msoFillSolid**

## **msoFillTextured**

*expression.Type*

*expression* Required. An expression that returns one of the above objects.

[Type property as it applies to the \*\*ColorFormat\*\* object.](#)

Returns or sets the color format type. Read-only **MsoColorType**.

MsoColorType can be one of these MsoColorType constants.

**msoColorTypeCMS**

**msoColorTypeCMYK**

**msoColorTypeInk**

**msoColorTypeMixed**

**msoColorTypeRGB**

**msoColorTypeScheme**

*expression.Type*

*expression* Required. An expression that returns a **ColorFormat** object.

[Type property as it applies to the \*\*ConnectorFormat\*\* object.](#)

Returns or sets the connector format type. Read/write **MsoConnectorType**.

MsoConnectorType can be one of these MsoConnectorType constants.

**msoConnectorCurve**

**msoConnectorElbow**

**msoConnectorStraight**

**msoConnectorTypeMixed**

*expression.Type*

*expression* Required. An expression that returns a **ConnectorFormat** object.

[Type property as it applies to the \*\*DataLabel\*\* and \*\*DataLabels\*\* objects.](#)

Returns or sets the label type. Read/write **VARIANT**.

*expression*.Type

*expression* Required. An expression that returns one of the above objects.

[Type property as it applies to the \*\*Diagram\*\* object.](#)

Returns or sets the diagram type. Read-only **MsoDiagramType**.

MsoDiagramType can be one of these MsoDiagramType constants.

**msoDiagramCycle**  
**msoDiagramMixed**  
**msoDiagramOrgChart**  
**msoDiagramPyramid**  
**msoDiagramRadial**  
**msoDiagramTarget**  
**msoDiagramVenn**

*expression*.Type

*expression* Required. An expression that returns a **Diagram** object.

[Type property as it applies to the \*\*ListDataFormat\*\* object.](#)

Indicates the data type of the list column. Read-only **XlListDataType**.

This property is used only for lists that are linked to a SharePoint site.

XlListDataType can be one of these XlListDataType constants.

**xlListDataTypeNone**  
**xlListDataTypeText**  
**xlListDataTypeMultiLineText**  
**xlListDataTypeMultiLineRichText**  
**xlListDataTypeCounter**  
**xlListDataTypeNumber**  
**xlListDataTypeCurrency**

**xListDataTypeDateTime**  
**xListDataTypeChoice**  
**xListDataTypeChoiceMulti**  
**xListDataTypeListLookup**  
**xListDataTypeCheckbox**  
**xListDataTypeHyperLink**

*expression.Type*

*expression* Required. An expression that returns a **ListDataFormat** object.

[Type property as it applies to the \*\*HPageBreak\*\* and \*\*VPageBreak\*\* objects.](#)

Returns or sets the page break type. Read/write [XIPageBreak](#).

XIPageBreak can be one of these XIPageBreak constants.

**xIPageBreakAutomatic**  
**xIPageBreakManual**  
**xIPageBreakNone**

*expression.Type*

*expression* Required. An expression that returns one of the above objects.

[Type property as it applies to the \*\*Parameter\*\* object.](#)

Returns or sets the parameter type. Read-only [XIParameterType](#).

XIParameterType can be one of these XIParameterType constants.

**xIConstant**  
**xIPrompt**  
**xIRange**

*expression.Type*

*expression* Required. An expression that returns a **Parameter** object.

 [Type property as it applies to the \*\*ShadowFormat\*\* object.](#)

Returns or sets the shadow format type. Read/write [MsoShadowType](#).

MsoShadowType can be one of these MsoShadowType constants.

**msoShadow1**

**msoShadow10**

**msoShadow11**

**msoShadow12**

**msoShadow13**

**msoShadow14**

**msoShadow15**

**msoShadow16**

**msoShadow17**

**msoShadow18**

**msoShadow19**

**msoShadow2**

**msoShadow20**

**msoShadow3**

**msoShadow4**

**msoShadow5**

**msoShadow6**

**msoShadow7**

**msoShadow8**

**msoShadow9**

**msoShadowMixed**

*expression.Type*

*expression* Required. An expression that returns a **ShadowFormat** object.

 [Type property as it applies to the \*\*Shape\*\* and \*\*ShapeRange\*\* objects.](#)

Returns or sets the shape type. Read-only [MsoShapeType](#).

MsoShapeType can be one of these MsoShapeType constants.

**msoAutoShape**

**msoCallout**

**msoCanvas**

**msoChart**

**msoComment**

**msoDiagram**

**msoEmbeddedOLEObject**

**msoFormControl**

**msoFreeform**

**msoGroup**

**msoLine**

**msoLinkedOLEObject**

**msoLinkedPicture**

**msoMedia** Can not be used with this property. This constant is used with shapes in other Microsoft Office applications.

**msoOLEControlObject**

**msoPicture**

**msoPlaceholder** Can not be used with this property. This constant is used with shapes in other Microsoft Office applications.

**msoScriptAnchor**

**msoShapeTypeMixed**

**msoTable**

**msoTextBox**

**msoTextEffect**

*expression*.**Type**

*expression* Required. An expression that returns one of the above objects.

[Type property as it applies to the SmartTagAction object.](#)

Returns an [XISmartTagControlType](#) that represents the type of Smart Document control displayed in the **Document Actions** task pane. Read-only **XISmartTagControlType**.

XlSmartTagControlType can be one of these XlSmartTagControlType constants.

**xlSmartTagControlActiveX**

**xlSmartTagControlButton**

**xlSmartTagControlCheckbox**

**xlSmartTagControlCombo**

**xlSmartTagControlHelp**

**xlSmartTagControlHelpURL**

**xlSmartTagControlImage**

**xlSmartTagControlLabel**

**xlSmartTagControlLink**

**xlSmartTagControlListbox**

**xlSmartTagControlRadioGroup**

**xlSmartTagControlSeparator**

**xlSmartTagControlSmartTag**

**xlSmartTagControlTextbox**

*expression.Type*

*expression* Required. An expression that returns a **Trendline** object.

[Type property as it applies to the \*\*Trendline\*\* object.](#)

Returns or sets the trendline type. Read/write [XlTrendlineType](#).

XlTrendlineType can be one of these XlTrendlineType constants.

**xlExponential**

**xlLinear**

**xlLogarithmic**

**xlMovingAvg**

**xlPolynomial**

**xlPower**

*expression.Type*

*expression* Required. An expression that returns a **Trendline** object.

[Type property as it applies to the \*\*Window\*\* object.](#)

Returns or sets the window type. Read-only [XlWindowType](#).

XlWindowType can be one of these XlWindowType constants.

**xlChartAsWindow**

**xlChartInPlace**

**xlClipboard**

**xlInfo**

**xlWorkbook**

*expression*.**Type**

*expression* Required. An expression that returns a **Window** object.

[Type property as it applies to the \*\*Worksheet\*\* object.](#)

Returns or sets the worksheet type. Read-only [XlSheetType](#).

XlSheetType can be one of these XlSheetType constants.

**xlChart**

**xlDialogSheet**

**xlExcel4IntlMacroSheet**

**xlExcel4MacroSheet**

**xlWorksheet**

*expression*.**Type**

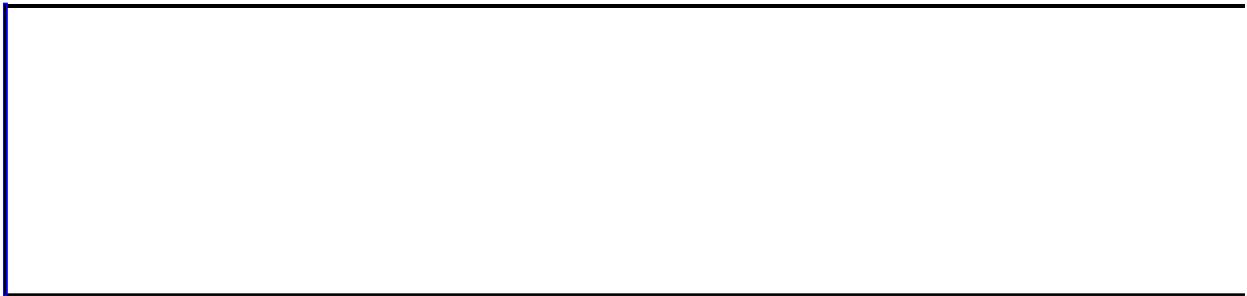
*expression* Required. An expression that returns a **Worksheet** object.

## Example

[As it applies to the \*\*Trendline\*\* object.](#)

This example changes the trendline type for the first series in embedded chart one on worksheet one. If the series has no trendline, this example fails.

```
Worksheets(1).ChartObjects(1).Chart _  
    .SeriesCollection(1).Trendlines(1).Type = xlMovingAvg
```



[Show All](#)

# Underline Property

Returns or sets the type of underline applied to the font. Can be one of the following [XlUnderlineStyle](#) constants. Read/write **Variant**.

XlUnderlineStyle can be one of these XlUnderlineStyle constants.

**xlUnderlineStyleNone**

**xlUnderlineStyleSingle**

**xlUnderlineStyleDouble**

**xlUnderlineStyleSingleAccounting**

**xlUnderlineStyleDoubleAccounting**

*expression*.**Underline**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example sets the font in the active cell on Sheet1 to single underline.

```
Worksheets("Sheet1").Activate  
ActiveCell.Font.Underline = xlUnderlineStyleSingle
```



# UnlockedFormulaCells Property

When set to **True** (default), Microsoft Excel identifies selected cells that are unlocked and contain a formula. **False** disables error checking for unlocked cells that contain formulas. Read/write **Boolean**.

*expression*.**UnlockedFormulaCells**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In the following example, the **AutoCorrect Options** button appears for cell A3, an unlocked cell containing a formula.

```
Sub CheckUnlockedCell()  
  
    Application.ErrorCheckingOptions.UnlockedFormulaCells = True  
    Range("A1").Value = 1  
    Range("A2").Value = 2  
    Range("A3").Formula = "=A1+A2"  
    Range("A3").Locked = False  
  
End Sub
```



# UpBars Property

Returns an [UpBars](#) object that represents the up bars on a line chart. Applies only to line charts. Read-only.

## Example

This example turns on up and down bars for chart group one in Chart1 and then sets their colors. The example should be run on a 2-D line chart containing two series that cross each other at one or more data points.

```
With Charts("Chart1").ChartGroups(1)
    .HasUpDownBars = True
    .DownBars.Interior.ColorIndex = 3
    .UpBars.Interior.ColorIndex = 5
End With
```



[Show All](#)

# UpdateLinks Property

Returns or sets an [XlUpdateLink](#) constant indicating a workbook's setting for updating embedded OLE links. Read/write.

XlUpdateLinks can be one of these XlUpdateLinks constants.

**xlUpdateLinksAlways** Embedded OLE links are always updated for the specified workbook.

**xlUpdateLinksNever** Embedded OLE links are never updated for the specified workbook.

**xlUpdateLinksUserSetting** Embedded OLE links are updated according to the user's settings for the specified workbook.

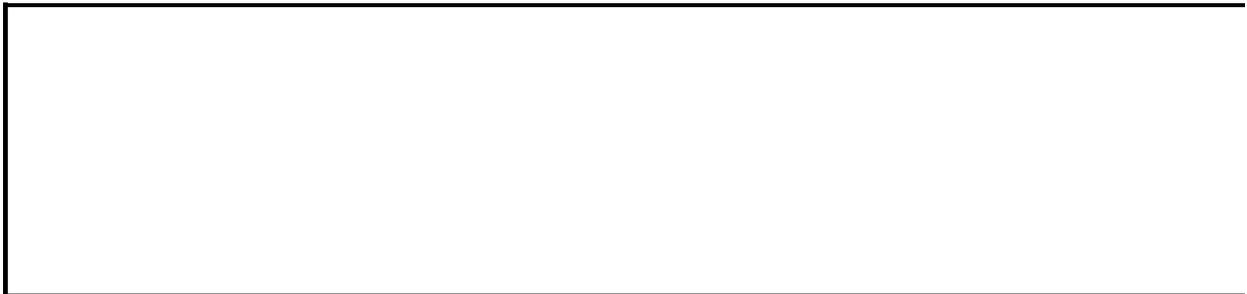
*expression*.**UpdateLinks**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel determines the setting for updating links and notifies the user.

```
Sub UseUpdateLinks()  
    Dim wkbOne As Workbook  
  
    Set wkbOne = Application.Workbooks(1)  
  
    Select Case wkbOne.UpdateLinks  
        Case xlUpdateLinksAlways  
            MsgBox "Links will always be updated " & _  
                "for the specified workbook."  
        Case xlUpdateLinksNever  
            MsgBox "Links will never be updated " & _  
                "for the specified workbook."  
        Case xlUpdateLinksUserSetting  
            MsgBox "Links will update according " & _  
                "to user setting for the specified workbook."  
    End Select  
  
End Sub
```



# UpdateLinksOnSave Property

**True** if hyperlinks and paths to all supporting files are automatically updated before you save the document as a Web page, ensuring that the links are up-to-date at the time the document is saved. **False** if the links are not updated. The default value is **True**. Read/write **Boolean**.

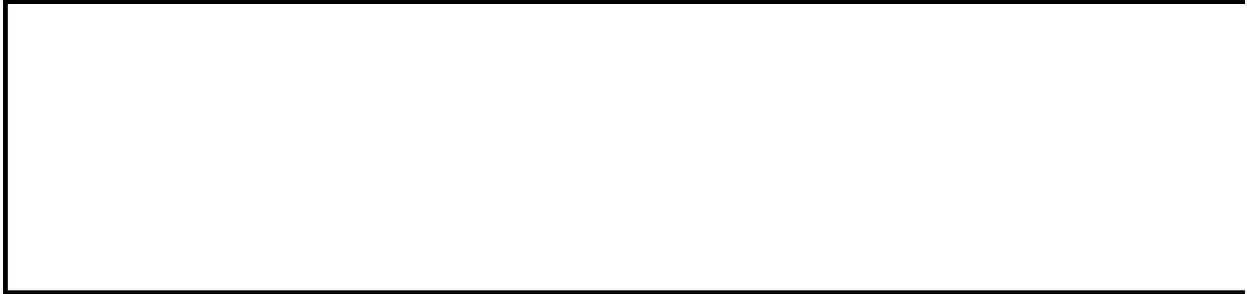
## Remarks

You should set this property to **False** if the location where the document is saved is different from the final location on the Web server and the supporting files are not available at the first location.

## Example

This example specifies that links are not updated before the document is saved.

```
Application.DefaultWebOptions.UpdateLinksOnSave = False
```



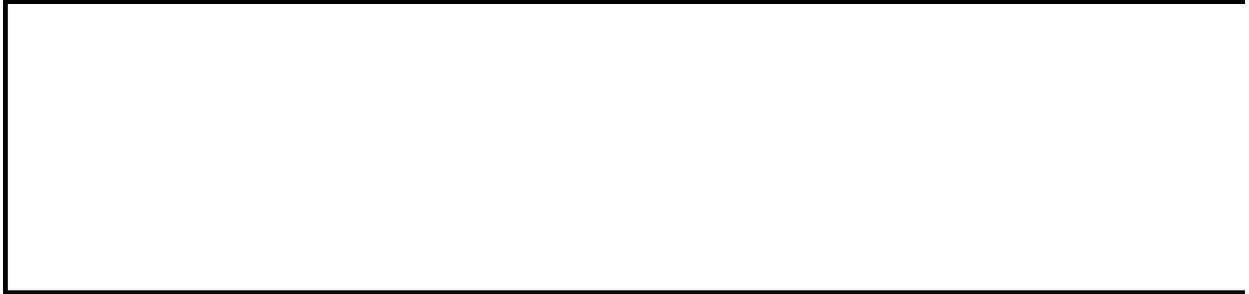
# UpdateRemoteReferences Property

**True** if Microsoft Excel updates remote references in for the workbook.  
Read/write **Boolean**.

## Example

This example causes remote references to be updated in the active workbook.

```
ActiveWorkbook.UpdateRemoteReferences = True
```



[Show All](#)

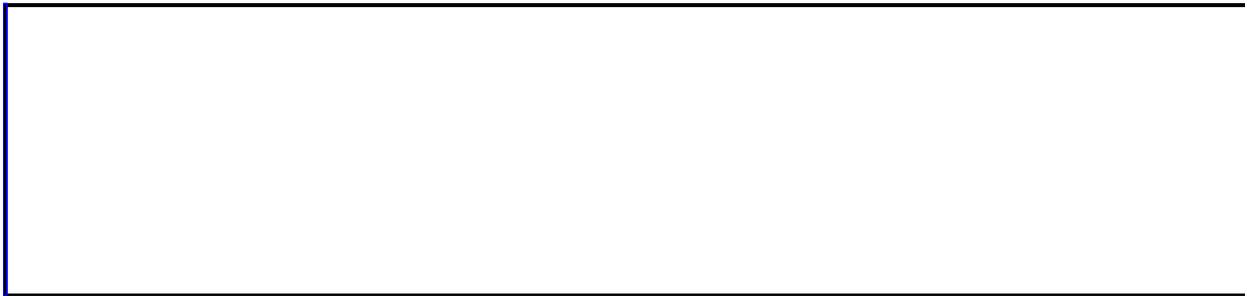
# Uri Property

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Returns a **String** that represents the [Uniform Resource Identifier \(URI\)](#) for the specified XML [namespace](#). Read-only.

*expression*.Uri

*expression* Required. An expression that returns an [XmlNamespace](#) object.



[Show All](#)

# UsableHeight Property

Returns the maximum height of the space that a window can occupy in the application window area, in [points](#). Read-only **Double**.

## Example

This example expands the active window to the maximum size available (assuming that the window isn't already maximized).

```
With ActiveWindow
    .WindowState = xlNormal
    .Top = 1
    .Left = 1
    .Height = Application.UsableHeight
    .Width = Application.UsableWidth
End With
```



[Show All](#)

# UsableWidth Property

Returns the maximum width of the space that a window can occupy in the application window area, in [points](#). Read-only **Double**.

## Example

This example expands the active window to the maximum size available (assuming that the window isn't already maximized).

```
With ActiveWindow
    .WindowState = xlNormal
    .Top = 1
    .Left = 1
    .Height = Application.UsableHeight
    .Width = Application.UsableWidth
End With
```



# UsedObjects Property

Returns a [UsedObjects](#) object representing objects allocated in a workbook.

*expression*.UsedObjects

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel determines the quantity of objects that have been allocated and notifies the user. This example assumes a recalculation was performed in the application and was interrupted before finishing.

```
Sub CountUsedObjects()
```

```
    MsgBox "The number of used objects in this application is: " & _  
        Application.UsedObjects.Count
```

```
End Sub
```



# UsedRange Property

Returns a [Range](#) object that represents the used range on the specified worksheet. Read-only.

## Example

This example selects the used range on Sheet1.

```
Worksheets("Sheet1").Activate  
ActiveSheet.UsedRange.Select
```



[Show All](#)

# UseLocalConnection Property

**True** if the [LocalConnection](#) property is used to specify the string that enables Microsoft Excel to connect to a data source. **False** if the connection string specified by the [Connection](#) property is used. Read/write **Boolean**.

## Example

This example sets the connection string of the first PivotTable cache to reference an [offline cube file](#).

```
With ActiveWorkbook.PivotCaches(1)
    .LocalConnection = _
        "OLEDB;Provider=MSOLAP;Data Source=C:\Data\DataCube.cub"
    .UseLocalConnection = True
End With
```



# UseLongFileNames Property

**True** if long file names are used when you save the document as a Web page.

**False** if long file names are not used and the DOS file name format (8.3) is used.

The default value is **True**. Read/write **Boolean**.

## Remarks

If you don't use long file names and your document has supporting files, Microsoft Excel automatically organizes those files in a separate folder. Otherwise, use the [OrganizeInFolder](#) property to determine whether supporting files are organized in a separate folder.

## Example

This example disallows the use of long file names as the global default for the application.

```
Application.DefaultWebOptions.UseLongFileNames = False
```



# UserControl Property

**True** if the application is visible or if it was created or started by the user. **False** if you created or started the application programmatically by using the **CreateObject** or **GetObject** functions, and the application is hidden. Read/write **Boolean**.

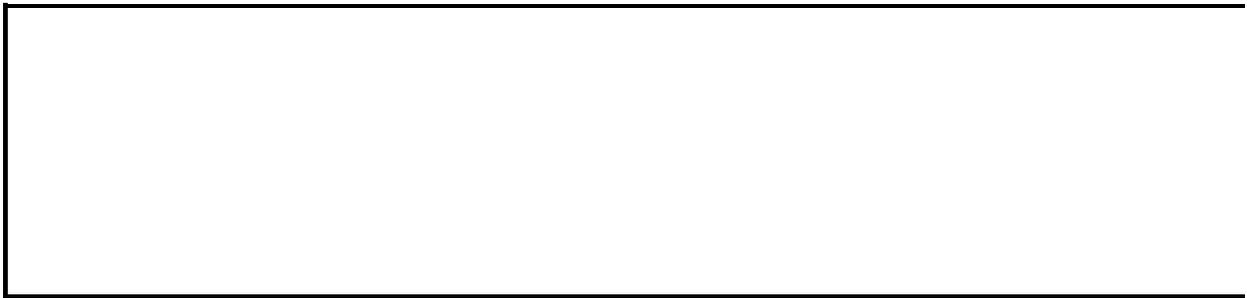
## Remarks

When the **UserControl** property is **False** for an object, that object is released when the last programmatic reference to the object is released. If this property is **False**, Microsoft Excel quits when the last object in the session is released.

## Example

This example displays the status of the **UserControl** property.

```
If Application.UserControl Then
    MsgBox "This workbook was created by the user"
Else
    MsgBox "This workbook was created programmatically"
End If
```



# UserDict Property

Instructs Microsoft Excel to create a custom dictionary to which new words can be added to, when performing spelling checks on a worksheet. Read/write **String**.

*expression*.**UserDict**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example instructs Microsoft Excel to create custom dictionary called "Custom1.dic" in the spelling options feature and notifies the user.

```
Sub SpecialWord()  
    Application.SpellingOptions.UserDict = "Custom1.dic"  
    MsgBox "The custom dictionary is currently set to: " _  
        & Application.SpellingOptions.UserDict  
End Sub
```



# UserLibraryPath Property

Returns the path to the location on the user's computer where the COM add-ins are installed. Read-only **String**.

## Example

This example determines where the COM add-ins are installed on the user's computer and assigns the string to the variable `strLibPath`.

```
strLibPath = Application.UserLibraryPath
```



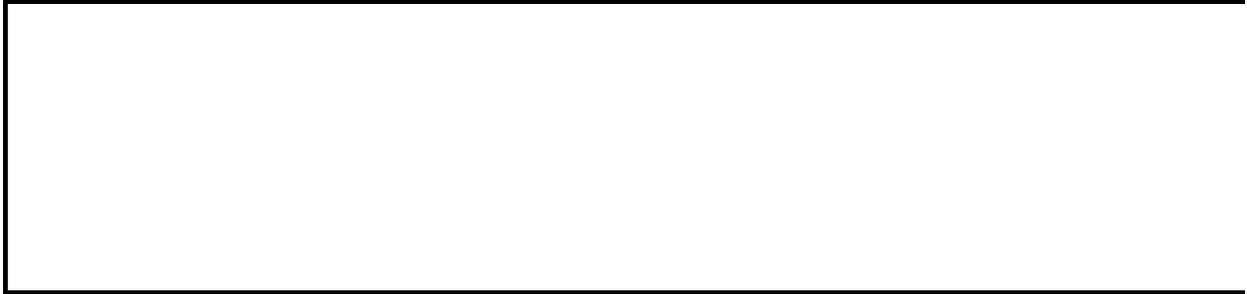
# UserName Property

Returns or sets the name of the current user. Read/write **String**.

## Example

This example displays the name of the current user.

```
MsgBox "Current user is " & Application.UserName
```



# Users Property

Returns a [UserAccessList](#) object for the protected range on a worksheet.

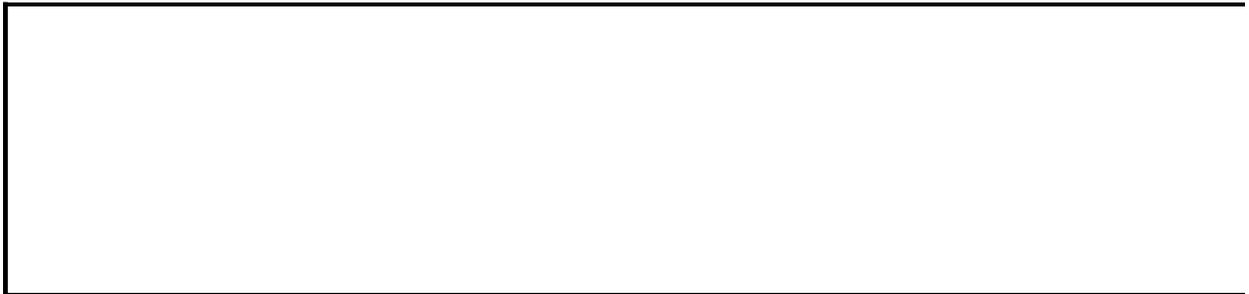
*expression*.Users

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel displays the name of the first user allowed access to the first protected range on the active worksheet. This example assumes that a range has been chosen to be protected and that a particular user has been given access to this range.

```
Sub DisplayUserName()  
    Dim wksSheet As Worksheet  
    Set wksSheet = Application.ActiveSheet  
    ' Display name of user with access to protected range.  
    MsgBox wksSheet.Protection.AllowEditRanges(1).Users(1).Name  
End Sub
```



# UserStatus Property

Returns a 1-based, two-dimensional array that provides information about each user who has the workbook open as a shared list. The first element of the second dimension is the name of the user, the second element is the date and time when the user last opened the workbook, and the third element is a number indicating the type of list (1 indicates exclusive, and 2 indicates shared). Read-only

**Variant.**

## Remarks

The **UserStatus** property doesn't return information about users who have the specified workbook open as read-only.

## Example

This example creates a new workbook and inserts into it information about all users who have the active workbook open as a shared list.

```
users = ActiveWorkbook.UserStatus
With Workbooks.Add.Sheets(1)
    For row = 1 To UBound(users, 1)
        .Cells(row, 1) = users(row, 1)
        .Cells(row, 2) = users(row, 2)
        Select Case users(row, 3)
            Case 1
                .Cells(row, 3).Value = "Exclusive"
            Case 2
                .Cells(row, 3).Value = "Shared"
        End Select
    Next
End With
```



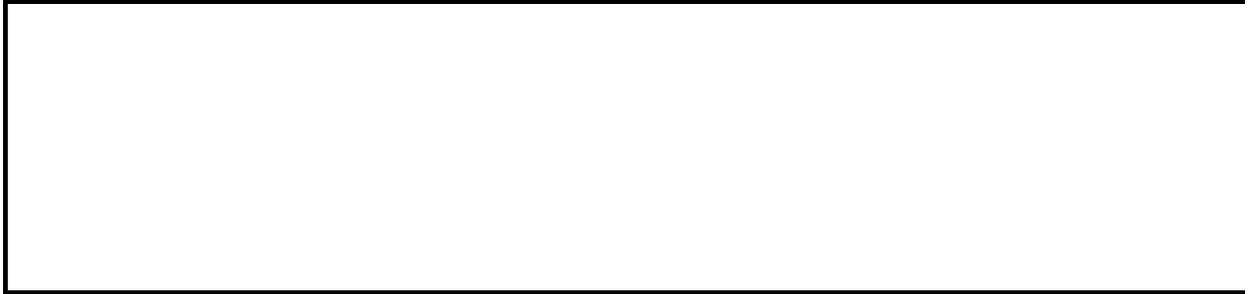
# UseStandardHeight Property

**True** if the row height of the **Range** object equals the standard height of the sheet. Returns **Null** if the range contains more than one row and the rows aren't all the same height. Read/write **Variant**.

## Example

This example sets the height of row one on Sheet1 to the standard height.

```
Worksheets("Sheet1").Rows(1).UseStandardHeight = True
```



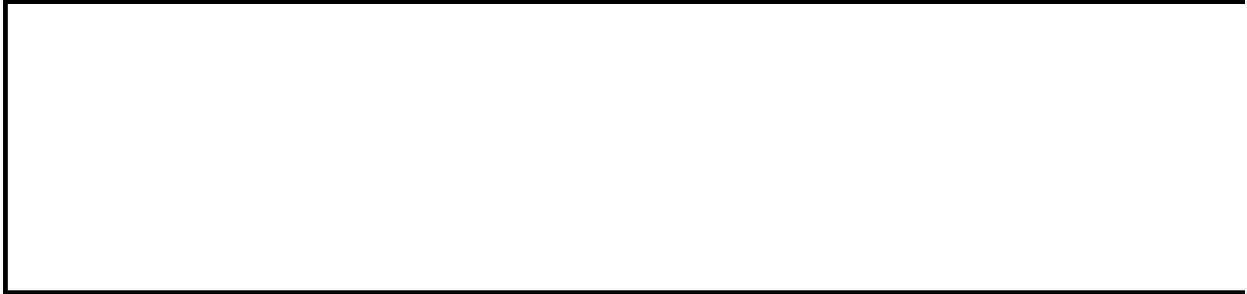
# UseStandardWidth Property

**True** if the column width of the **Range** object equals the standard width of the sheet. Returns **Null** if the range contains more than one column and the columns aren't all the same width. Read/write **Variant**.

## Example

This example sets the width of column A on Sheet1 to the standard width.

```
Worksheets("Sheet1").Columns("A").UseStandardWidth = True
```



# UseSystemSeparators Property

**True** (default) if the system separators of Microsoft Excel are enabled.  
Read/write **Boolean**.

*expression*.**UseSystemSeparators**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, "1,234,567.89" is placed in cell A1. The system separators are then changed to dashes for the decimals and thousands separators.

```
Sub ChangeSystemSeparators()  
  
    Range("A1").Formula = "1,234,567.89"  
    MsgBox "The system separators will now change."  
  
    ' Define separators and apply.  
    Application.DecimalSeparator = "-"  
    Application.ThousandsSeparator = "-"  
    Application.UseSystemSeparators = False  
  
End Sub
```



# VacatedStyle Property

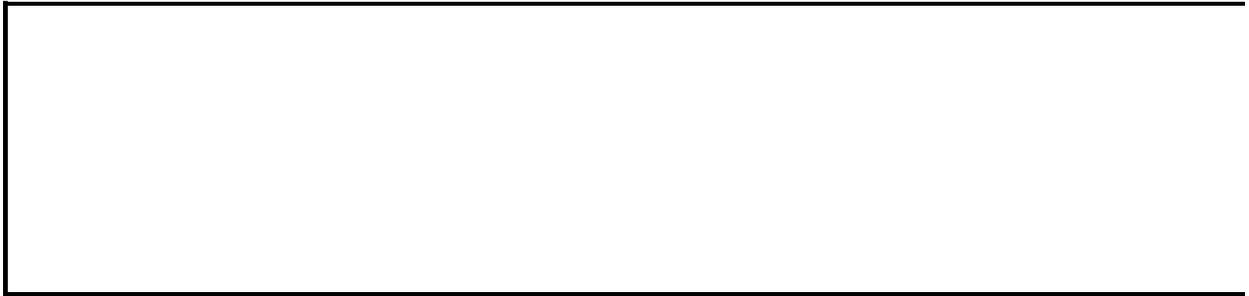
Returns or sets the style applied to cells vacated when the PivotTable report is refreshed. The default value is a null string (no style is applied by default).

Read/write **String**.

## Example

This example sets the vacated cells in the PivotTable report to the BlackAndBlue style.

```
Worksheets(1).PivotTables("Pivot1").VacatedStyle = "BlackAndBlue"
```



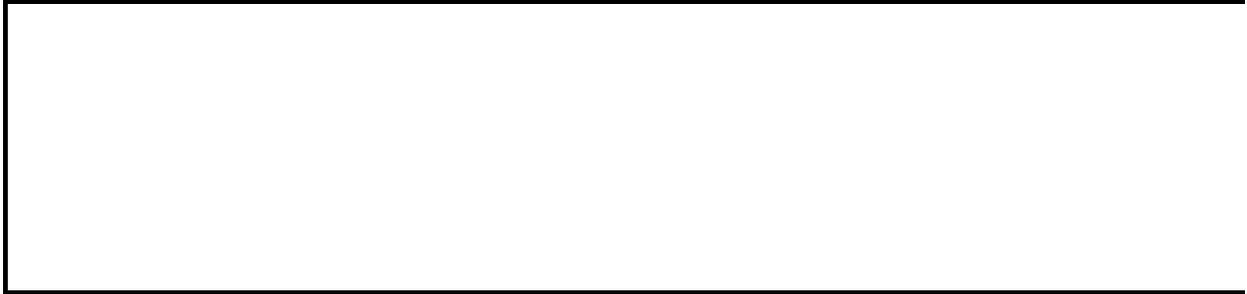
# Validation Property

Returns the [Validation](#) object that represents data validation for the specified range. Read-only.

## Example

This example causes data validation for cell E5 to allow blank values.

```
Range("e5").Validation.IgnoreBlank = True
```



[Show All](#)

# Value Property

[Value property as it applies to the \*\*Application\*\*, \*\*CubeField\*\*, and \*\*Style\*\* objects.](#)

For the **Application** object, always returns "Microsoft Excel". For the **CubeField** object, the name of the specified field. For the **Style** object, the name of the specified style. Read-only **String**.

*expression*.**Value**

*expression* Required. An expression that returns one of the above objects.

[Value property as it applies to the \*\*Borders\*\* and \*\*CustomProperty\*\* objects.](#)

Synonym for `Borders.LineStyle`. Read/write **Variant**.

*expression*.**Value**

*expression* Required. An expression that returns one of the above objects.

[Value property as it applies to the \*\*ControlFormat\*\* object.](#)

The name of specified control format. Read/write **Long**.

*expression*.**Value**

*expression* Required. An expression that returns a **ControlFormat** object.

[Value property as it applies to the \*\*Error\*\* and \*\*Validation\*\* objects.](#)

**True** if all the validation criteria are met (that is, if the range contains valid data). Read-only **Boolean**.

*expression*.**Value**

*expression* Required. An expression that returns one of the above objects.

[Value property as it applies to the \*\*Name\*\*, \*\*PivotField\*\*, \*\*PivotFormula\*\*, \*\*PivotItem\*\*, and \*\*PivotTable\*\* objects.](#)

For the **Name** object, a string containing the formula that the name is defined to refer to. The string is in A1-style notation in the language of the macro, and it begins with an equal sign. For the **PivotField** object, the name of the specified field in the PivotTable report. For the **PivotFormula** object, the name of the specified formula in the PivotTable formula. For the **PivotItem** object, the name of the specified item in the PivotTable field. For the PivotTable object, the name of the PivotTable report. Read/write **String**.

*expression*.**Value**

*expression* Required. An expression that returns one of the above objects.

[Value property as it applies to the \*\*Parameter\*\* object.](#)

The parameter value. For more information, see the **Parameter** object. Read-only **Variant**.

*expression*.**Value**

*expression* Required. An expression that returns a **Parameter** object.

[Value property as it applies to the \*\*Range\*\* object.](#)

Returns or sets the value of the specified range. Read/write **Variant**.

*expression*.**Value**(**RangeValueDataType**)

*expression* Required. An expression that returns a **Range** object.

**RangeValueDataType** Optional **Variant**. The range value data type. Can be a [xlRangeValueDataType](#) constant.

xlRangeValueDataType can be one of these xlRangeValueDataType constants.

**xlRangeValueDefault** *default* If the specified **Range** object is empty, returns the value **Empty** (use the **IsEmpty** function to test for this case). If the **Range** object contains more than one cell, returns an array of values (use the **IsArray**

function to test for this case).

**xlRangeValueMSPersistXML** Returns the recordset representation of the specified **Range** object in an XML format.

**xlRangeValueXMLSpreadsheet** Returns the values, formatting, formulas and names of the specified **Range** object in the XML Spreadsheet format.

## Remarks

When setting a range of cells with the contents of an XML spreadsheet file, only values of the first sheet in the workbook are used. You cannot set or get a discontinuous range of cells in the XML spreadsheet format.



[Value property as it applies to the \*\*XmlNamespaces\*\* object.](#)

Returns a **String** that represents the XML namespaces that have been added to the workbook.

*expression*.**Value()**

*expression* Required. An expression that returns an **XmlNamespaces** object.

## Remarks

If the workbook contains more than one namespace, then the namespaces are separated by a blank space in the string returned by the **Value** property.

[Value property as it applies to the XPath object.](#)

Returns a **String** that represents the XPath for the specified object.

*expression*.**Value()**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

[As it applies to the \*\*Range\*\* object.](#)

This example sets the value of cell A1 on Sheet1 to 3.14159.

```
Worksheets("Sheet1").Range("A1").Value = 3.14159
```

This example loops on cells A1:D10 on Sheet1. If one of the cells has a value less than 0.001, the code replaces the value with 0 (zero).

```
For Each c in Worksheets("Sheet1").Range("A1:D10")  
    If c.Value < .001 Then  
        c.Value = 0  
    End If  
Next c
```



# Value2 Property

Returns or sets the cell value. Read/write **Variant**.

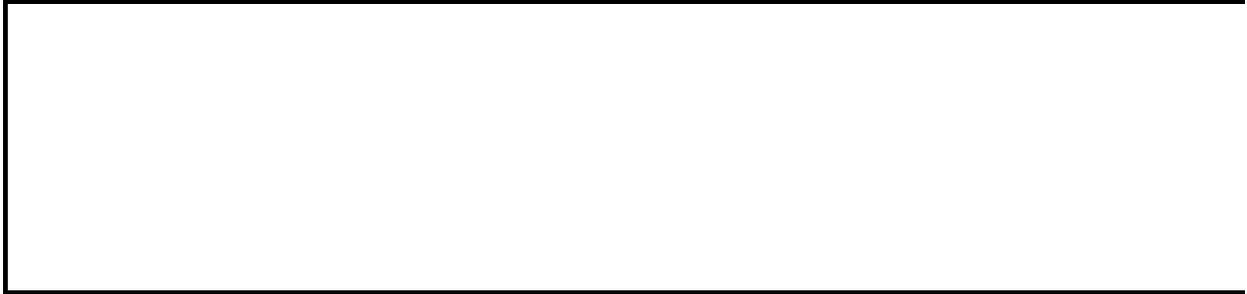
## Remarks

The only difference between this property and the **Value** property is that the **Value2** property doesn't use the **Currency** and **Date** data types. You can return values formatted with these data types as floating-point numbers by using the **Double** data type.

## Example

This example uses the **Value2** property to add the values of two cells.

```
Range("a1").Value2 = Range("b1").Value2 + Range("c1").Value2
```



[Show All](#)

# Values Property

 [Values property as it applies to the \*\*Scenario\*\* object.](#)

Returns an array that contains the current values of the changing cells for the scenario. Read-only **Variant**.

*expression*.**Values**(*Index*)

*expression* Required. An expression that returns one of the above objects.

**Index** Optional **Variant**. The position of the value.

 [Values property as it applies to the \*\*Series\*\* object.](#)

Returns or sets a collection of all the values in the series. This can be a range on a worksheet or an array of constant values, but not a combination of both. See the examples for details. Read/write **Variant**.

*expression*.**Values**

*expression* Required. An expression that returns one of the above objects.

## **Remark**

For PivotChart reports, this property is read-only.

## Example

This example sets the series values from a range.

```
Charts("Chart1").SeriesCollection(1).Values = _  
    Worksheets("Sheet1").Range("C5:T5")
```

To assign a constant value to each individual data point, you must use an array.

```
Charts("Chart1").SeriesCollection(1).Values = _  
    Array(1, 3, 5, 7, 11, 13, 17, 19)
```



# VaryByCategories Property

**True** if Microsoft Excel assigns a different color or pattern to each data marker. The chart must contain only one series. Read/write **Boolean**.

## Example

This example assigns a different color or pattern to each data marker in chart group one. The example should be run on a 2-D line chart that has data markers on a series.

```
Charts("Chart1").ChartGroups(1).VaryByCategories = True
```



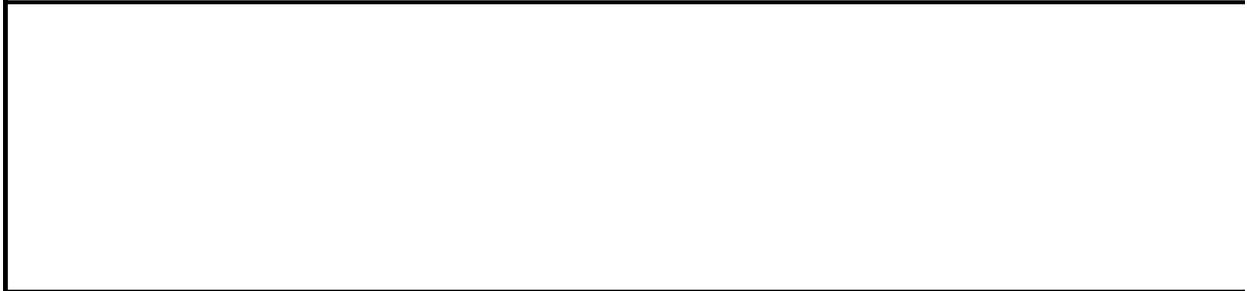
# VBA Signed Property

**True** if the Visual Basic for Applications project for the specified workbook has been digitally signed. Read-only **Boolean**.

## Example

This example loads a workbook named “mybook.xls” and then tests to see whether its Visual Basic for Applications project has a digital signature. If there’s no digital signature, the example displays a warning message.

```
Workbooks.Open FileName:="c:\My Documents\mybook.xls", _  
    ReadOnly:=False  
If Workbook.VBASigned = False Then  
    MsgBox "Warning! The project "  
        "has not been digitally signed." _  
        , vbCritical, "Digital Signature Warning"  
End If
```



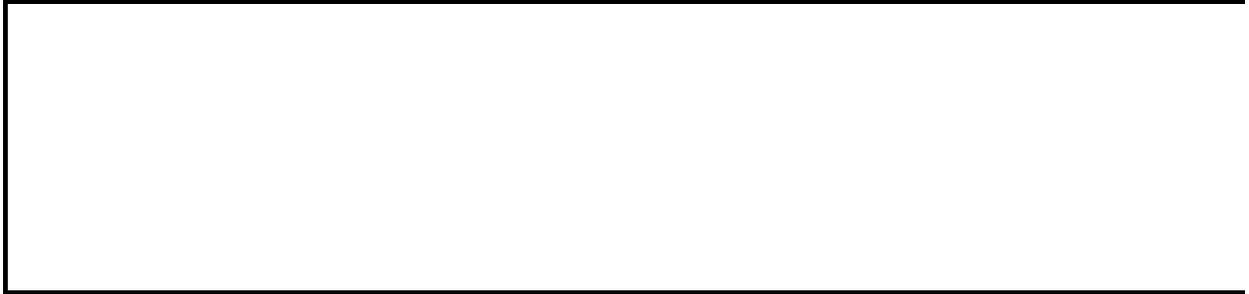
# VBE Property

Returns a **VBE** object that represents the Visual Basic Editor. Read-only.

## Example

This example changes the name of the active Visual Basic project.

```
Application.VBE.ActiveVBProject.Name = "TestProject"
```



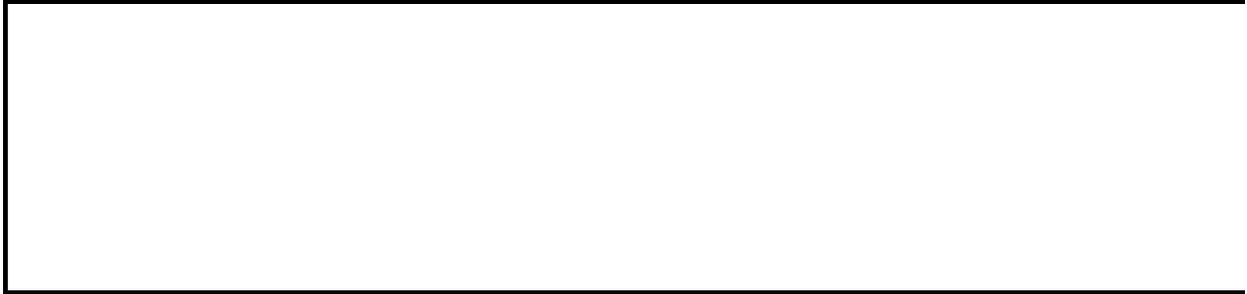
# **VBProject Property**

Returns a **VBProject** object that represents the Visual Basic project in the specified workbook. Read-only.

## Example

This example changes the name of the Visual Basic project in the workbook.

```
ThisWorkbook.VBProject.Name = "TestProject"
```



[Show All](#)

# Version Property

 [Version property as it applies to the \*\*PivotTable\*\* object.](#)

Returns the Microsoft Excel version number. Read-only **[XlPivotTableVersionList](#)**.

XlPivotTableVersionList can be one of these XlPivotTableVersionList constants.

**xlPivotTableVersion10**

**xlPivotTableVersion2000**

**xlPivotTableVersionCurrent**

*expression*.**Version**

*expression* Required. An expression that returns one of the above objects.

 [Version property as it applies to the \*\*Application\*\* object.](#)

Returns the Microsoft Excel version number. Read-only **String**.

*expression*.**Version**

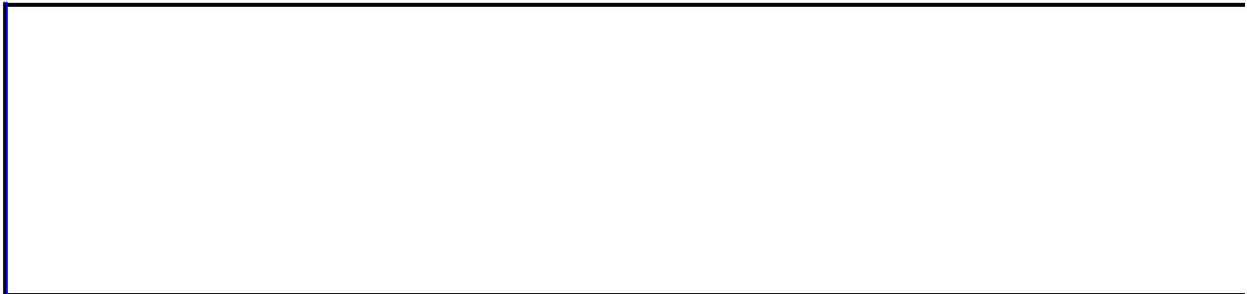
*expression* Required. An expression that returns one of the above objects.

## Example

 [As it applies to the \*\*Application\*\* object.](#)

This example displays a message box that contains the Microsoft Excel version number and the name of the operating system.

```
MsgBox "Welcome to Microsoft Excel version " & _  
    Application.Version & " running on " & _  
    Application.OperatingSystem & "!"
```



[Show All](#)

# VerticalAlignment Property

 [VerticalAlignment property as it applies to the \*\*Style\*\* and \*\*TextFrame\*\* objects.](#)

Returns or sets the vertical alignment of the specified object. Read/write [XIVAlign](#).

XIVAlign can be one of these XIVAlign constants.

**xIVAlignCenter**

**xIVAlignJustify**

**xIVAlignBottom**

**xIVAlignDistributed**

**xIVAlignTop**

*expression*.**VerticalAlignment**

*expression* Required. An expression that returns one of the above objects.

## Remarks

Some of these constants may not be available to you, depending on the language support (U.S. English, for example) that you've selected or installed.

 [VerticalAlignment](#) property as it applies to the [AxisTitle](#), [CellFormat](#), [ChartTitle](#), [DataLabel](#), [DataLabels](#), [DisplayUnitLabel](#), and [Range](#) objects.

Returns or sets the vertical alignment of the specified object. Read/write  
**Variant**.

*expression*.**VerticalAlignment**

*expression* Required. An expression that returns one of the above objects.

## Example

 [As it applies to the \*\*CellFormat\*\* object.](#)

This example sets the height of row 2 on Sheet1 to twice the standard height and then centers the contents of the row vertically.

```
Worksheets("Sheet1").Rows(2).RowHeight = _  
    2 * Worksheets("Sheet1").StandardHeight  
Worksheets("Sheet1").Rows(2).VerticalAlignment = xlVAlignCenter
```



[Show All](#)

# VerticalFlip Property

**True** if the specified shape is flipped around the vertical axis. Read-only [MsoTriState](#).

MsoTriState can be one of these MsoTriState constants.

**msoCTrue**

**msoFalse**

**msoTriStateMixed**

**msoTriStateToggle**

**msoTrue** The specified shape is flipped around the vertical axis.

*expression*.**VerticalFlip**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example restores each shape on myDocument to its original state if it's been flipped horizontally or vertically.

```
Set myDocument = Worksheets(1)
For Each s In myDocument.Shapes
    If s.HorizontalFlip Then s.Flip msoFlipHorizontal
    If s.VerticalFlip Then s.Flip msoFlipVertical
Next
```



[Show All](#)

# Vertices Property

Returns the coordinates of the specified freeform drawing's vertices (and control points for Bézier curves) as a series of [coordinate pairs](#). You can use the array returned by this property as an argument to the [AddCurve](#) method or [AddPolyLine](#) method. Read-only **Variant**.

The following table shows how the **Vertices** property associates the values in the array `vertArray()` with the coordinates of a triangle's vertices.

<b>vertArray element</b>	<b>Contains</b>
<code>vertArray(1, 1)</code>	The horizontal distance from the first vertex to the left side of the document
<code>vertArray(1, 2)</code>	The vertical distance from the first vertex to the top of the document
<code>vertArray(2, 1)</code>	The horizontal distance from the second vertex to the left side of the document
<code>vertArray(2, 2)</code>	The vertical distance from the second vertex to the top of the document
<code>vertArray(3, 1)</code>	The horizontal distance from the third vertex to the left side of the document
<code>vertArray(3, 2)</code>	The vertical distance from the third vertex to the top of the document

## Example

This example assigns the vertex coordinates for shape one on myDocument to the array variable vertArray() and displays the coordinates for the first vertex.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes(1)
    vertArray = .Vertices
    x1 = vertArray(1, 1)
    y1 = vertArray(1, 2)
    MsgBox "First vertex coordinates: " & x1 & ", " & y1
End With
```

This example creates a curve that has the same geometric description as shape one on myDocument. Shape one must contain  $3n+1$  vertices for this example to succeed.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes
    .AddCurve .Item(1).Vertices
End With
```



[Show All](#)

# View Property

Returns or sets the view showing in the window. Read/write [XLWindowView](#).

XLWindowView can be one of these XLWindowView constants.

**xlNormalView**

**xlPageBreakPreview**

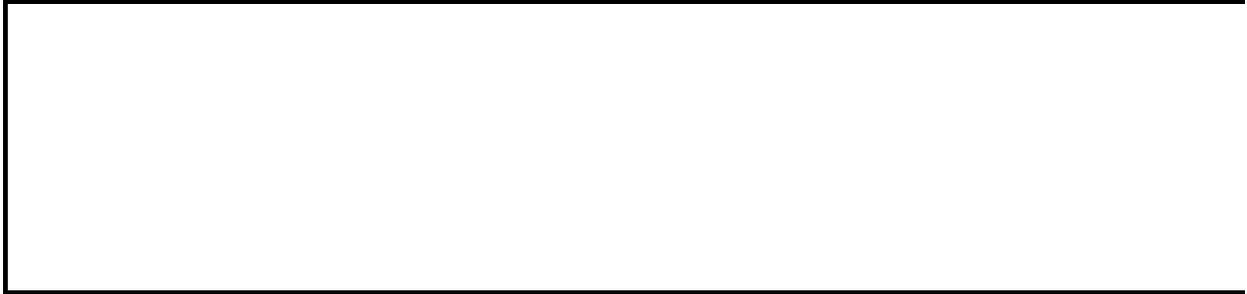
*expression*.**View**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example switches the view in the active window to page break preview.

```
ActiveWindow.View = xlPageBreakPreview
```



[Show All](#)

# ViewCalculatedMembers Property

When set to **True** (default), calculated members for [Online Analytical Processing \(OLAP\)](#) PivotTables can be viewed. Read/write **Boolean**.

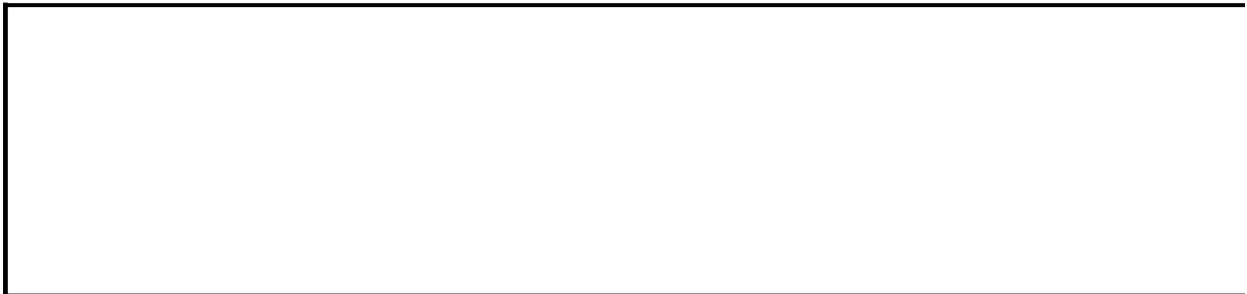
*expression*.**ViewCalculatedMembers**

*expression* Required. An expression that returns a [PivotTable](#) object.

## Example

This example determines if calculated members can be viewed on the PivotTable and notifies the user. It assumes that a PivotTable exists on the active worksheet.

```
Sub CheckViewCalculatedMembers()  
    Dim pvtTable As PivotTable  
  
    Set pvtTable = ActiveSheet.PivotTables(1)  
  
    ' Determine if calculated members can be viewed.  
    If pvtTable.ViewCalculatedMembers = True Then  
        MsgBox "Calculated members can be viewed."  
    Else  
        MsgBox "Calculated members cannot be viewed."  
    End If  
  
End Sub
```



[Show All](#)

# Visible Property

 [Visible property as it applies to the \*\*ChartFillFormat\*\*, \*\*FillFormat\*\*, \*\*LineFormat\*\*, \*\*ShadowFormat\*\*, \*\*Shape\*\*, \*\*ShapeRange\*\*, and \*\*ThreeDFormat\*\* objects.](#)

Determines whether the object is visible. Read/write [MsoTriState](#).

MsoTriState can be one of these MsoTriState constants.

**msoCTrue**

**msoFalse**

**msoTriStateMixed**

**msoTriStateToggle**

**msoTrue** The object is visible.

*expression.Visible*

*expression* Required. An expression that returns one of the above objects.

 [Visible property as it applies to the \*\*Chart\*\* and \*\*Worksheet\*\* objects.](#)

Determines whether the object is visible. Read/write [XlSheetVisibility](#).

XlSheetVisibility can be one of these XlSheetVisibility constants.

**xlSheetHidden**

**xlSheetVisible**

**xlSheetVeryHidden** Hides the object so that the only way for you to make it visible again is by setting this property to **True** (the user cannot make the object visible).

*expression.Visible*

*expression* Required. An expression that returns one of the above objects.

 [Visible property as it applies to the \*\*Application\*\*, \*\*ChartObject\*\*,](#)

[ChartObjects, Comment, Name, OLEObject, OLEObjects, Phonetic, Phonetics, PivotItem, and Window](#) objects.

Determines whether the object is visible. Read/write **Boolean**.

*expression*. **Visible**

*expression* Required. An expression that returns one of the above objects.

[Visible property as it applies to the Charts, Sheets, and Worksheets](#) objects.

Determines whether the object is visible. Read/write **Variant**.

*expression*. **Visible**

*expression* Required. An expression that returns one of the above objects.

## Remarks

The **Visible** property for a PivotTable item is **True** if the item is currently visible in the table.

If you set the **Visible** property for a name to **False**, the name won't appear in the **Define Name** dialog box.

## Example

 [As it applies to the \*\*Charts\*\*, \*\*Sheets\*\*, and \*\*Worksheets\*\* objects.](#)

This example hides Sheet1.

```
Worksheets("Sheet1").Visible = False
```

This example makes Sheet1 visible.

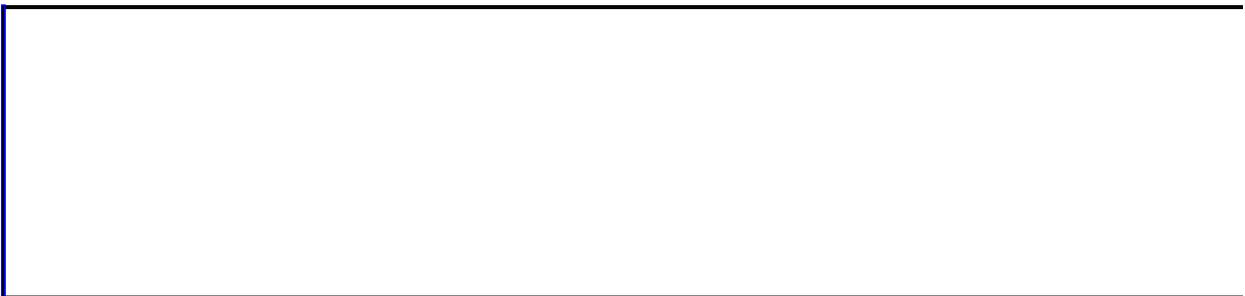
```
Worksheets("Sheet1").Visible = True
```

This example makes every sheet in the active workbook visible.

```
For Each sh In Sheets  
    sh.Visible = True  
Next sh
```

This example creates a new worksheet and then sets its **Visible** property to **xlVeryHidden**. To refer to the sheet, use its object variable, `newSheet`, as shown in the last line of the example. To use the `newSheet` object variable in another procedure, you must declare it as a public variable (`Public newSheet As Object`) in the first line of the module preceding any **Sub** or **Function** procedure.

```
Set newSheet = Worksheets.Add  
newSheet.Visible = xlVeryHidden  
newSheet.Range("A1:D4").Formula = "=RAND()"
```



[Show All](#)

# VisibleFields Property

Returns an object that represents either a single field in a PivotTable report (a [PivotField](#) object) or a collection of all the visible fields (a [PivotFields](#) object). Visible fields are shown as row, column, page or data fields. Read-only.

*expression*.VisibleFields(*Index*)

*expression* Required. An expression that returns a **PivotTable** object.

**Index** Optional **Variant**. The name or number of the field to be returned (can be an array to specify more than one field).

## Remarks

For [OLAP](#) data sources, there are no hidden fields, and this property returns all the fields in the PivotTable cache.

## Example

This example adds the visible field names to a list on a new worksheet.

```
Set nwSheet = Worksheets.Add
nwSheet.Activate
Set pvtTable = Worksheets("Sheet2").Range("A1").PivotTable
rw = 0
For Each pvtField In pvtTable.VisibleFields
    rw = rw + 1
    nwSheet.Cells(rw, 1).Value = pvtField.Name
Next pvtField
```



[Show All](#)

# VisibleItems Property

Returns an object that represents either a single visible PivotTable item (a [PivotItem](#) object) or a collection of all the visible items (a [PivotItems](#) object) in the specified field. Read-only.

*expression*.VisibleItems(*Index*)

*expression* Required. An expression that returns a **PivotField** object.

**Index** Optional **Variant**. The number or name of the item to be returned (can be an array to specify more than one item).

## Remarks

For [OLAP](#) data sources, this property is read-only and always returns **True**. There are no hidden items.

## Example

This example adds the names of all visible items in the field named "Product" to a list on a new worksheet.

```
Set nwSheet = Worksheets.Add
nwSheet.Activate
Set pvtTable = Worksheets("Sheet2").Range("A1").PivotTable
rw = 0
For Each pvtItem In pvtTable.PivotFields("Product").VisibleItems
    rw = rw + 1
    nwSheet.Cells(rw, 1).Value = pvtItem.Name
Next
```



# VisibleRange Property

Returns a [Range](#) object that represents the range of cells that are visible in the window or pane. If a column or row is partially visible, it's included in the range. Read-only.

## Example

This example displays the number of cells visible on Sheet1.

```
Worksheets("Sheet1").Activate  
MsgBox "There are " & Windows(1).VisibleRange.Cells.Count _  
    & " cells visible"
```



[Show All](#)

# VisualTotals Property

**True** (default) to enable [Online Analytical Processing \(OLAP\)](#) PivotTables to retotal after an item has been hidden from view. Read/write **Boolean**.

*expression*.**VisualTotals**

*expression* Required. An expression that returns a [PivotTable](#) object.

## Remarks

In non-OLAP PivotTables, if you hide an item, the total is recomputed to reflect only the items that remain visible in the PivotTable. In an OLAP PivotTable, the total is computed on the server and is therefore unaffected by whether any items are hidden in the PivotTable view. However, if the **VisualTotals** property is set to **False** for an OLAP PivotTable, then the results of the OLAP PivotTable will match those of the non-OLAP PivotTable.

For OLAP PivotTables, a **VisualTotals** property setting of **True** (default) works the same way as described for non-OLAP PivotTables.

The **VisualTotals** property returns **True** for all new PivotTables. However, if you open a workbook in the current version of Microsoft Excel and the PivotTable had been created in a previous version of Excel, then the **VisualTotals** property will return **False**.

**Note** All previously created PivotTables will have the **VisualTotals** property set to **False** by default, unless the user changes it, but for all newly created ones the **VisualTotals** property is set to **True**.

## Example

This example determines if the ability to re-total after an item has been hidden from view is available for OLAP PivotTables and notifies the user. The example assumes a PivotTable exists on the active worksheet.

```
Sub CheckVisualTotals()  
  
    Dim pvtTable As PivotTable  
  
    Set pvtTable = ActiveSheet.PivotTables(1)  
  
    ' Determine if visual totals is enabled for OLAP PivotTables.  
    If pvtTable.VisualTotals = True Then  
        MsgBox "Ability enabled to re-total after an item " & _  
            "has been hidden from view."  
    Else  
        MsgBox "Unable to re-total items not hidden from view."  
    End If  
  
End Sub
```



# VPageBreaks Property

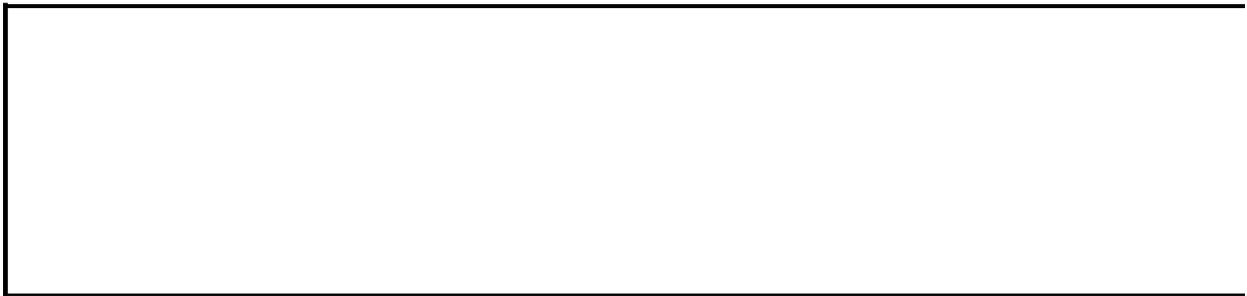
Returns a [VPageBreaks](#) collection that represents the vertical page breaks on the sheet. Read-only.

For information about returning a single member of a collection, see [Returning an Object from a Collection](#).

## Example

This example displays the total number of full-screen and print-area vertical page breaks.

```
For Each pb in Worksheets(1).VPPageBreaks
    If pb.Extent = xlPageBreakFull Then
        cFull = cFull + 1
    Else
        cPartial = cPartial + 1
    End If
Next
MsgBox cFull & " full-screen page breaks, " & cPartial & _
    " print-area page breaks"
```



# Walls Property

Returns a [Walls](#) object that represents the walls of the 3-D chart. Read-only.

## Remarks

This property doesn't apply to 3-D pie charts.

## Example

This example sets the color of the wall border of Chart1 to red. The example should be run on a 3-D chart.

```
Charts("Chart1").Walls.Border.ColorIndex = 3
```



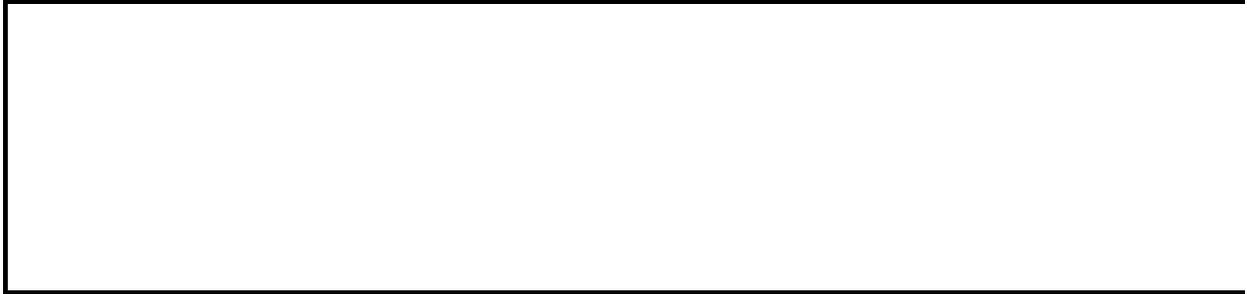
# WallsAndGridlines2D Property

**True** if gridlines are drawn two-dimensionally on a 3-D chart. Read/write **Boolean**.

## Example

This example causes Microsoft Excel to draw 2-D gridlines on Chart1.

```
Charts("Chart1").WallsAndGridlines2D = True
```



# Watches Property

Returns a [Watches](#) object representing a range which is tracked when the worksheet is recalculated.

*expression*.**Watches**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example creates a summation formula in cell A3, and then adds this cell to the Watch Window.

```
Sub AddWatch()  
    With Application  
        .Range("A1").Formula = 1  
        .Range("A2").Formula = 2  
        .Range("A3").Formula = "=Sum(A1:A2)"  
        .Range("A3").Select  
        .Watches.Add Source:=ActiveCell  
    End With  
End Sub
```



# WebConsecutiveDelimitersAsOne Property

**True** if consecutive delimiters are treated as a single delimiter when you import data from HTML <PRE> tags in a Web page into a query table, and if the data is to be parsed into columns. **False** if you want to treat consecutive delimiters as multiple delimiters. The default value is **True**. Read/write **Boolean**.

## Remarks

Use this property only when the query table's [QueryType](#) property is set to **xlWebQuery**, the query returns an HTML document, and the [WebPreFormattedTextToColumns](#) property is set to **True**.

## Example

This example sets the space character to be the delimiter in the query table on the first worksheet in the first workbook, and then it refreshes the query table. Consecutive spaces are treated as a single space.

```
Set shFirstQtr = Workbooks(1).Worksheets(1)
Set qtQtrResults = shFirstQtr.QueryTables _
    .Add(Connection := "URL;http://datasvr/98q1/19980331.htm", _
        Destination := shFirstQtr.Cells(1,1))
With qtQtrResults
    .WebConsecutiveDelimitersAsOne = True
    .Refresh
End With
```



# WebDisableDateRecognition Property

**True** if data that resembles dates is parsed as text when you import a Web page into a query table. **False** if date recognition is used. The default value is **False**.  
Read/write **Boolean**.

## Remarks

Use this property only when the query table's [QueryType](#) property is set to **xlWebQuery** and the query returns an HTML document.

## Example

This example turns off date recognition so that Web page data that resembles dates is imported as text. The example then refreshes the query table.

```
Set shFirstQtr = Workbooks(1).Worksheets(1)
Set qtQtrResults = shFirstQtr.QueryTables _
    .Add(Connection := "URL;http://datasvr/98q1/19980331.htm", _
        Destination := shFirstQtr.Cells(1,1))
With qtQtrResults
    .WebDisableDateRecognition = True
    .Refresh
End With
```



# WebDisableRedirections Property

**True** if Web query redirections are disabled for a [QueryTable](#) object. The default value is **False**. Read/write **Boolean**.

*expression*.**WebDisableRedirections**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, Microsoft Excel determines the settings of Web query redirections for the first worksheet in the workbook. This example assumes a **QueryTable** object exists on the first worksheet, otherwise a run-time error will occur.

```
Sub CheckWebQuerySetting()  
    Dim wksSheet As Worksheet  
    Set wksSheet = Application.ActiveSheet  
    MsgBox wksSheet.QueryTables(1).WebDisableRedirections  
End Sub
```



[Show All](#)

# WebFormatting Property

Returns or sets a value that determines how much formatting from a Web page, if any, is applied when you import the page into a query table. Read/write [XIWebFormatting](#).

XIWebFormatting can be one of these XIWebFormatting constants.

**xlWebFormattingAll**

**xlWebFormattingRTF**

**xlWebFormattingNone** *default.*

*expression*.**WebFormatting**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Use this property only when the query table's [QueryType](#) property is set to **xlWebQuery** and the query returns an HTML document.

## Example

This example adds a new Web query table to the first worksheet in the first workbook, imports all of the Web page formatting applied to the data, and then refreshes the query table.

```
Set shFirstQtr = Workbooks(1).Worksheets(1)
Set qtQtrResults = shFirstQtr.QueryTables _
    .Add(Connection := "URL;http://datasvr/98q1/19980331.htm", _
        Destination := shFirstQtr.Cells(1,1))
With qtQtrResults
    .WebFormatting = xlAll
    .Refresh
End With
```



# WebOptions Property

Returns the [WebOptions](#) collection, which contains workbook-level attributes used by Microsoft Excel when you save a document as a Web page or open a Web page. Read-only.

## Example

This example specifies that cascading style sheets and Western document encoding be used when items in the first workbook are saved to a Web page.

```
Set objW0 = Workbooks(1).WebOptions  
objW0.RelyOnCSS = True  
objW0.Encoding = msoEncodingWestern
```



# WebPreFormattedTextToColumns Property

Returns or sets whether data contained within HTML <PRE> tags in the Web page is parsed into columns when you import the page into a query table. The default is **True**. Read/write **Boolean**.

## Remarks

This property is used only when the [QueryType](#) property of the query table is **xlWebQuery** and the query returns a HTML document.

## Example

This example adds a new Web query table to the first worksheet in the first workbook. Note that the example doesn't parse into columns any data located between the HTML <PRE> tags.

```
Set shFirstQtr = Workbooks(1).Worksheets(1)
Set qtQtrResults = shFirstQtr.QueryTables _
    .Add(Connection := "URL;http://datasvr/98q1/19980331.htm", _
        Destination := shFirstQtr.Cells(1,1))
With qtQtrResults
    .WebFormatting = xlNone
    .WebPreFormattedTextToColumns = False
    .Refresh
End With
```



[Show All](#)

# WebSelectionType Property

Returns or sets a value that determines whether an entire Web page, all tables on the Web page, or only specific tables on the Web page are imported into a query table. Read/write [XIWebSelectionType](#).

XIWebSelectionType can be one of these XIWebSelectionType constants.

**xlEntirePage**

**xlAllTables** *default.*

**xlSpecifiedTables**

*expression*.**WebSelectionType**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

Use this property only when the query table's [QueryType](#) property is set to **xlWebQuery** and the query returns an HTML document.

If the value of this property is **xlSpecifiedTables**, you can use the [WebTables](#) property to specify the tables to be imported.

## Example

This example adds a new Web query table to the first worksheet in the first workbook and then imports data from the first and second tables in the Web page.

```
Set shFirstQtr = Workbooks(1).Worksheets(1)
Set qtQtrResults = shFirstQtr.QueryTables _
    .Add(Connection := "URL;http://datasvr/98q1/19980331.htm", _
        Destination := shFirstQtr.Cells(1,1))
With qtQtrResults
    .WebFormatting = xlNone
    .WebSelectionType = xlSpecifiedTables
    .WebTables = "1,2"
    .Refresh
End With
```



# WebSingleBlockTextImport Property

**True** if data from the HTML <PRE> tags in the specified Web page is processed all at once when you import the page into a query table. **False** if the data is imported in blocks of contiguous rows so that header rows will be recognized as such. The default value is **False**. Read/write **Boolean**.

## Remarks

Use this property only when the query table's [QueryType](#) property is set to **xlWebQuery** and the query returns an HTML document.

## Example

This example adds a new Web query table to the first worksheet in the first workbook and then imports all of the HTML <PRE> tag data all at once.

```
Set shFirstQtr = Workbooks(1).Worksheets(1)
Set qtQtrResults = shFirstQtr.QueryTables _
    .Add(Connection := "URL;http://datasvr/98q1/19980331.htm", _
        Destination := shFirstQtr.Cells(1,1))
With qtQtrResults
    .WebSingleBlockTextImport = True
    .Refresh
End With
```



# WebTables Property

Returns or sets a comma-delimited list of table names or table index numbers when you import a Web page into a query table. Read/write **String**.

## Remarks

Use this property only when the query table's [QueryType](#) property is set to **xlWebQuery**, the query returns an HTML document, and the value of the [WebSelectionType](#) property is **xlSpecifiedTables**.

## Example

This example adds a new Web query table to the first worksheet in the first workbook and then imports data from the first and second tables in the Web page.

```
Set shFirstQtr = Workbooks(1).Worksheets(1)
Set qtQtrResults = shFirstQtr.QueryTables _
    .Add(Connection := "URL;http://datasvr/98q1/19980331.htm", _
        Destination := shFirstQtr.Cells(1,1))
With qtQtrResults
    .WebFormatting = xlNone
    .WebSelectionType = xlSpecifiedTables
    .WebTables = "1,2"
    .Refresh
End With
```



[Show All](#)

# Weight Property

 [Weight property as it applies to the \*\*LineFormat\*\* object.](#)

Returns or sets the weight of the line. Read/write **Single**.

*expression*.**Weight**

*expression* Required. An expression that returns one of the above objects.

 [Weight property as it applies to the \*\*Border\*\* and \*\*Borders\*\* objects.](#)

Returns or sets the weight of the border. Read/write **XlBorderWeight**.

XlBorderWeight can be one of these XlBorderWeight constants.

**xlHairline**

**xlThin**

**xlMedium**

**xlThick**

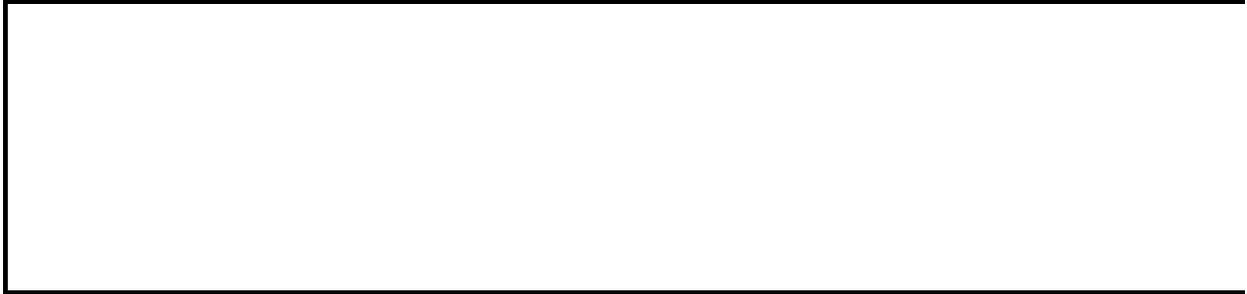
*expression*.**Weight**

*expression* Required. An expression that returns one of the objects.

## Example

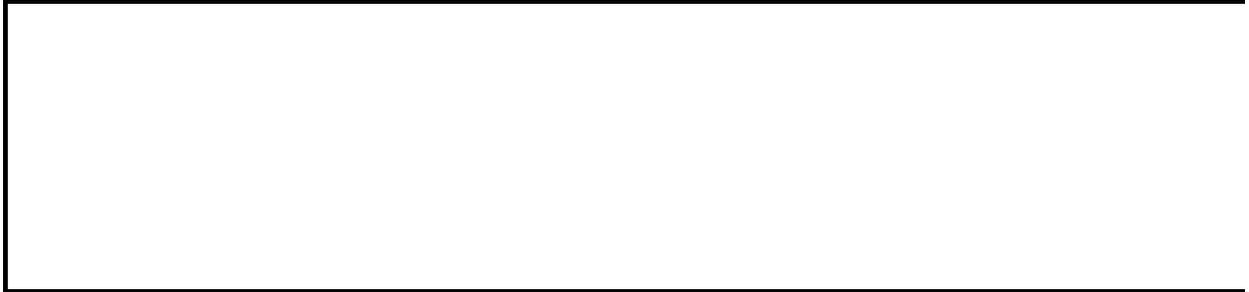
This example sets the border weight for oval one on Sheet1.

```
Worksheets("Sheet1").Ovals(1).Border.Weight = xlMedium
```



# WhichAddress Property

You have requested Help for a Visual Basic keyword used only on the Macintosh. For information about this keyword, consult the language reference Help included with Microsoft Office Macintosh Edition.



[Show All](#)

# Width Property

 [Width property as it applies to the \*\*Application\*\* object.](#)

The distance from the left edge of the application window to its right edge. If the window is minimized, `Application.Width` is read-only and returns the width of the window icon. Read/write **Double**.

*expression.Width*

*expression* Required. An expression that returns an [Application](#) object.

 [Width property as it applies to the \*\*Window\*\* object.](#)

The width of the window. Use the [UsableWidth](#) property to determine the maximum size for the window. You cannot set this property if the window is maximized or minimized. Use the [WindowState](#) property to determine the window state. Read/write **Double**.

*expression.Width*

*expression* Required. An expression that returns a [Window](#) object.

 [Width property as it applies to the \*\*ChartArea\*\*, \*\*ChartObject\*\*, \*\*ChartObjects\*\*, \*\*Legend\*\*, \*\*OLEObject\*\*, \*\*OLEObjects\*\*, and \*\*PlotArea\*\* objects.](#)

The width of the object. Read/write **Double**.

*expression.Width*

*expression* Required. An expression that returns one of the above objects.

 [Width property as it applies to the \*\*Axis\*\*, \*\*LegendEntry\*\*, and \*\*LegendKey\*\* objects.](#)

The width of the object. Read-only **Double**.

*expression*.**Width**

*expression* Required. An expression that returns one of the above objects.

 [Width property as it applies to the \*\*Graphic\*\*, \*\*Shape\*\*, and \*\*ShapeRange\*\* objects.](#)

The width of the object. Read/write **Single**.

*expression*.**Width**

*expression* Required. An expression that returns one of the above objects.

 [Width property as it applies to the \*\*Range\*\* object.](#)

The width of the range. Read-only **Variant**.

*expression*.**Width**

*expression* Required. An expression that returns a **Range** object.

## Example

 [As it applies to the \*\*Application\*\* object.](#)

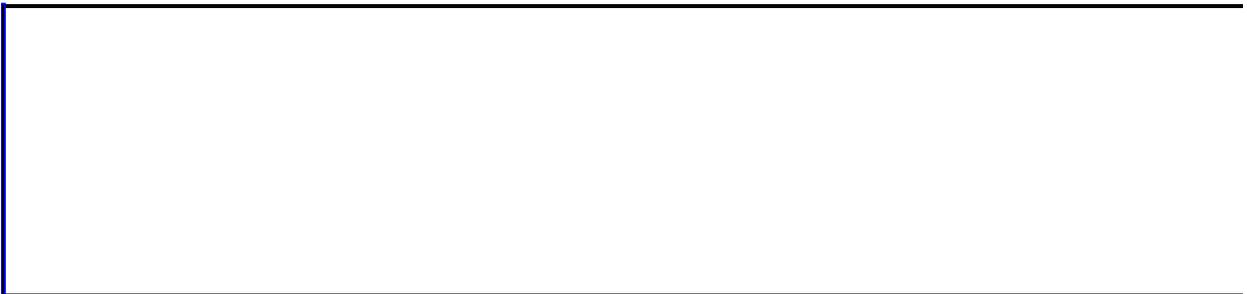
This example expands the active window to the maximum size available (assuming that the window isn't maximized).

```
With ActiveWindow
    .WindowState = xlNormal
    .Top = 1
    .Left = 1
    .Height = Application.UsableHeight
    .Width = Application.UsableWidth
End With
```

 [As it applies to the \*\*ChartArea\*\*, \*\*ChartObject\*\*, \*\*ChartObjects\*\*, \*\*Legend\*\*, \*\*OLEObject\*\*, \*\*OLEObjects\*\*, and \*\*PlotArea\*\* objects.](#)

This example sets the width of the embedded chart.

```
Worksheets("Sheet1").ChartObjects(1).Width = 360
```



# WindowNumber Property

Returns the window number. For example, a window named "Book1.xls:2" has 2 as its window number. Most windows have the window number 1. Read-only **Long**.

## Remarks

The window number isn't the same as the window index (the return value of the **Index** property), which is the position of the window within the **Windows** collection.

## Example

This example creates a new window of the active window and then displays the window number of the new window.

```
ActiveWindow.NewWindow  
MsgBox ActiveWindow.WindowNumber
```



# Windows Property

For an **Application** object, returns a [Windows](#) collection that represents all the windows in all the workbooks. For a **Workbook** object, returns a **Windows** collection that represents all the windows in the specified workbook. Read-only **Windows** object.

## Remarks

For information about returning a single member of a collection, see [Returning an Object from a Collection](#).

Using this property without an object qualifier is equivalent to using `Application.Windows`.

This property returns a collection of both visible and hidden windows.

## Example

This example closes the first open or hidden window in Microsoft Excel.

```
Application.Windows(1).Close
```

This example names window one in the active workbook "Consolidated Balance Sheet." This name is then used as the index to the **Windows** collection.

```
ActiveWorkbook.Windows(1).Caption = "Consolidated Balance Sheet"  
ActiveWorkbook.Windows("Consolidated Balance Sheet") _  
    .ActiveSheet.Calculate
```



# WindowsForPens Property

**True** if the computer is running under Microsoft Windows for Pen Computing.  
Read-only **Boolean**.

## Example

This example shows how to limit handwriting recognition to numbers and punctuation only if Microsoft Windows for Pen Computing is running.

```
If Application.WindowsForPens Then  
    Application.ConstrainNumeric = True  
End If
```



[Show All](#)

# WindowState Property

Returns or sets the state of the window. Read/write [XIWindowState](#).

XIWindowState can be one of these XIWindowState constants.

**xlMaximized**

**xlNormal**

**xlMinimized**

*expression*.**WindowState**

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

This example maximizes the application window in Microsoft Excel.

```
Application.WindowState = xlMaximized
```

This example expands the active window to the maximum size available (assuming that the window isn't already maximized).

```
With ActiveWindow  
    .WindowState = xlNormal  
    .Top = 1  
    .Left = 1  
    .Height = Application.UsableHeight  
    .Width = Application.UsableWidth  
End With
```



# Workbooks Property

Returns a [Workbooks](#) collection that represents all the open workbooks. Read-only.

For information about returning a single member of a collection, see [Returning an Object from a Collection](#).

## Remarks

Using this property without an object qualifier is equivalent to using `Application.Workbooks`.

The collection returned by the **Workbooks** property doesn't include open add-ins, which are a special kind of hidden workbook. You can, however, return a single open add-in if you know the file name. For example, `workbooks("Oscar.xla")` will return the open add-in named "Oscar.xla" as a **Workbook** object.

## Example

This example activates the workbook Book1.xls.

```
Workbooks("BOOK1").Activate
```

This example opens the workbook Large.xls.

```
Workbooks.Open filename:="LARGE.XLS"
```

This example saves changes to and closes all workbooks except the one that's running the example.

```
For Each w In Workbooks  
    If w.Name <> ThisWorkbook.Name Then  
        w.Close savechanges:=True  
    End If  
Next w
```



# Worksheet Property

Returns a [Worksheet](#) object that represents the worksheet containing the specified range. Read-only.

## Example

This example displays the name of the worksheet that contains the active cell. The example must be run from a worksheet.

```
MsgBox ActiveCell.Worksheet.Name
```

This example displays the name of the worksheet that contains the range named "testRange."

```
MsgBox Range("testRange").Worksheet.Name
```



# WorksheetFunction Property

Returns the [WorksheetFunction](#) object. Read-only.

## Example

This example displays the result of applying the **Min** worksheet function to the range A1:A10.

```
Set myRange = Worksheets("Sheet1").Range("A1:C10")  
answer = Application.WorksheetFunction.Min(myRange)  
MsgBox answer
```



# Worksheets Property

For an **Application** object, returns a [Sheets](#) collection that represents all the worksheets in the active workbook. For a **Workbook** object, returns a [Sheets](#) collection that represents all the worksheets in the specified workbook. Read-only **Sheets** object.

## Remarks

For information about returning a single member of a collection, see [Returning an Object from a Collection](#).

Using this property without an object qualifier returns all the worksheets in the active workbook.

This property doesn't return macro sheets; use the [Excel4MacroSheets](#) property or the [Excel4IntlMacroSheets](#) property to return those sheets.

## Example

This example displays the value in cell A1 on Sheet1 in the active workbook.

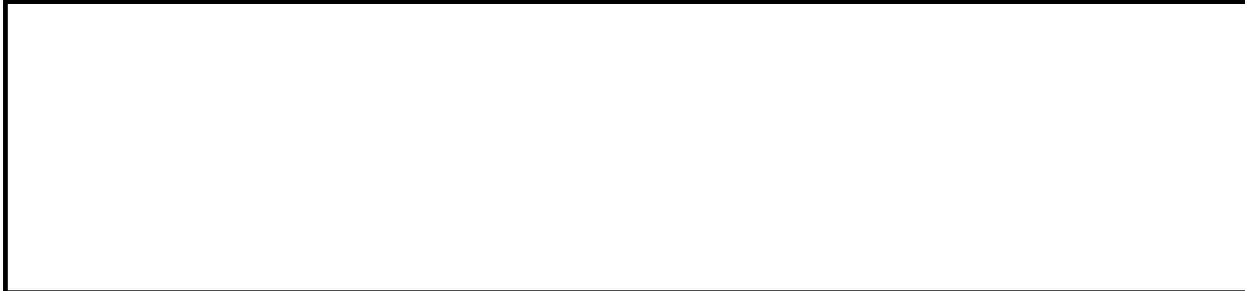
```
MsgBox worksheets("Sheet1").Range("A1").Value
```

This example displays the name of each worksheet in the active workbook.

```
For Each ws In worksheets  
    MsgBox ws.Name  
Next ws
```

This example adds a new worksheet to the active workbook and then sets the name of the worksheet.

```
Set newSheet = worksheets.Add  
newSheet.Name = "current Budget"
```



[Show All](#)

# WrapText Property

 [WrapText property as it applies to the \*\*Style\*\* object.](#)

**True** if Microsoft Excel wraps the text in the object. Read/write **Boolean**.

*expression*.**WrapText**

*expression* Required. An expression that returns one of the above objects.

 [WrapText property as it applies to the \*\*CellFormat\*\* and \*\*Range\*\* objects.](#)

**True** if Microsoft Excel wraps the text in the object. Returns **Null** if the specified range contains some cells that wrap text and other cells that don't. Read/write **Variant**.

*expression*.**WrapText**

*expression* Required. An expression that returns one of the above objects.

## **Remarks**

Microsoft Excel will change the row height of the range, if necessary, to accommodate the text in the range.

## Example

 [As it applies to the \*\*Range\*\* object.](#)

This example formats cell B2 on Sheet1 so that the text wraps within the cell.

```
Worksheets("Sheet1").Range("B2").Value = _  
    "This text should wrap in a cell."  
Worksheets("Sheet1").Range("B2").WrapText = True
```



# WritePassword Property

Returns or sets a **String** for the write password of a workbook. Read/write.

*expression*.**WritePassword**

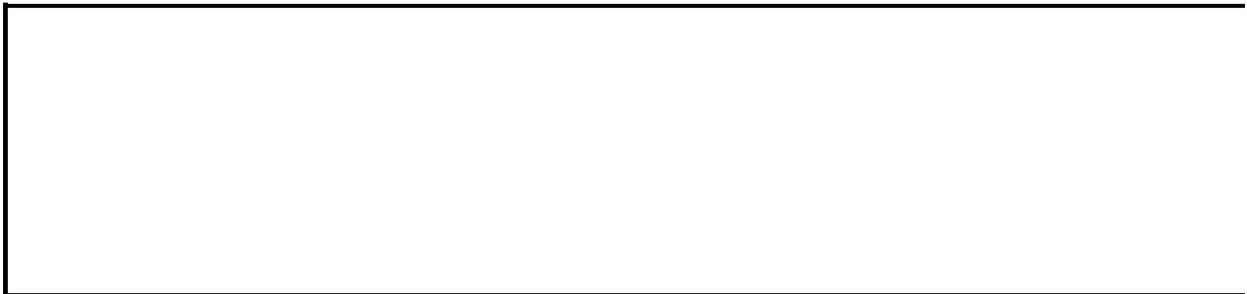
*expression* Required. An expression that returns one of the objects in the Applies To list.

## Example

In this example, if the active workbook is not protected against saving changes, Microsoft Excel sets the password to a string as the write password for the active workbook.

```
Sub UseWritePassword()  
  
    Dim strPassword As String  
  
    strPassword = InputBox ("Enter the password")  
  
    ' Set password to a string if allowed.  
    If ActiveWorkbook.WriteReserved = False Then  
        ActiveWorkbook.WritePassword = strPassword  
    End If  
  
End Sub
```

**Note** The **WritePassword** property is readable and returns ">>\*\*\*".



# WriteReserved Property

**True** if the workbook is write-reserved. Read-only **Boolean**.

## Remarks

Use the [SaveAs](#) method to set this property.

## Example

If the active workbook is write-reserved, this example displays a message that contains the name of the user who saved the workbook as write-reserved.

```
With ActiveWorkbook
    If .WriteReserved = True Then
        MsgBox "Please contact " & .WriteReservedBy & Chr(13) & _
            " if you need to insert data in this workbook."
    End If
End With
```



# WriteReservedBy Property

Returns the name of the user who currently has write permission for the workbook. Read-only **String**.

## Example

If the active workbook is write-reserved, this example displays a message that contains the name of the user who saved the workbook as write-reserved.

```
With ActiveWorkbook
    If .WriteReserved = True Then
        MsgBox "Please contact " & .WriteReservedBy & Chr(13) & _
            " if you need to insert data in this workbook."
    End If
End With
```



[Show All](#)

# XML Property

 [XML property as it applies to the \*\*SmartTag\*\* object.](#)

Returns a **String** representing a sample of the XML that would be passed to the action handler. Read-only.

*expression.XML*

*expression* Required. An expression that returns a **SmartTag** object.

 [XML property as it applies to the \*\*XmlSchema\*\* object.](#)

Returns a **String** representing the content of the specified schema.

*expression.XML*

*expression* Required. An expression that returns a **XmlSchema** object.

## Example

 [As it applies to the SmartTag object.](#)

This example adds a smart tag to cell A1 and then displays the XML that would be passed to the action handler. This example assumes the host system is connected to the Internet.

```
Sub CheckXML()  
  
    Dim strLink As String  
    Dim strType As String  
  
    ' Define SmartTag variables.  
    strLink = "urn:schemas-microsoft-com:smarctags#StockTickerSymbol"  
    strType = "stockview"  
  
    ' Enable smart tags to be embedded and recognized.  
    ActiveWorkbook.SmartTagOptions.EmbedSmartTags = True  
    Application.SmartTagRecognizers.Recognize = True  
  
    Range("A1").Formula = "MSFT"  
  
    ' Display the sample of the XML.  
    MsgBox Range("A1").SmartTags.Add(strLink).XML  
  
End Sub
```



# XmlMap Property

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Returns an **XmlMap** object that represents the schema map used for the specified list. Read-only.

*expression*.**XmlMap**

*expression* Required. An expression that returns a [ListObject](#) object.



# XmlMaps Property

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Returns an **XmlMaps** collection that represents the schema maps that have been added to the specified workbook. Read-only.

*expression.XmlMaps*

*expression* Required. An expression that returns a [Workbook](#) object.



# XmlNamespaces Property

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Returns an **XmlNamespaces** collection that represents the XML namespaces contained in the specified workbook. Read-only.

*expression*.**XmlNamespaces**

*expression* Required. An expression that returns a [Workbook](#) object.



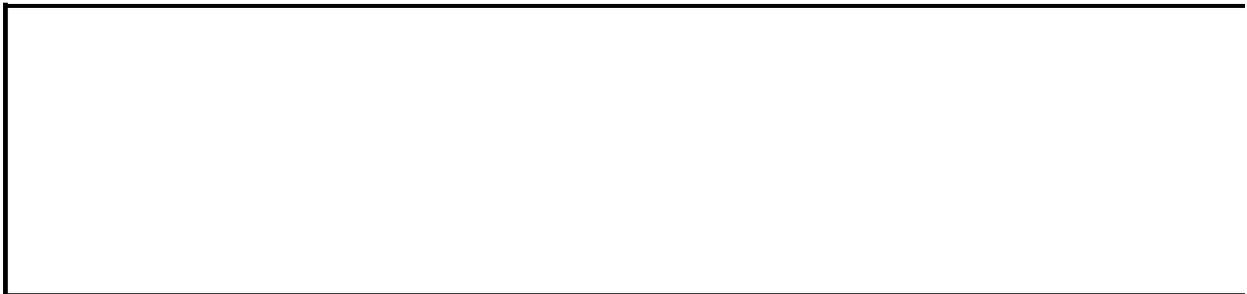
# XPath Property

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Returns an **XPath** object that represents the Xpath of the element mapped to the specified **Range** object. Read-only.

*expression*.**XPath**

*expression* Required. An expression that returns one of the objects in the Applies To list.



# XValues Property

Returns or sets an array of x values for a chart series. The **XValues** property can be set to a range on a worksheet or to an array of values, but it cannot be a combination of both. Read/write **Variant**.

For PivotChart reports, this property is read-only.

## Example

This example sets the x values for series one in Chart1 to the range B1:B5 on Sheet1.

```
Charts("Chart1").SeriesCollection(1).XValues = _  
    Worksheets("Sheet1").Range("B1:B5")
```

This example uses an array to set values for the individual points in series one in Chart1.

```
Charts("Chart1").SeriesCollection(1).XValues = _  
    Array(5.0, 6.3, 12.6, 28, 50)
```



[Show All](#)

# Zoom Property

 [Zoom property as it applies to the \*\*PageSetup\*\* object.](#)

Returns or sets a percentage (between 10 and 400 percent) by which Microsoft Excel will scale the worksheet for printing. Applies only to worksheets.  
Read/write **Variant**.

*expression*.**Zoom**

*expression* Required. An expression that returns a **PageSetup** object.

## Remarks

If this property is **False**, the [FitToPagesWide](#) and [FitToPagesTall](#) properties control how the worksheet is scaled.

All scaling retains the aspect ratio of the original document.



[Zoom property as it applies to the \*\*Window\*\* object.](#)

Returns or sets the display size of the window, as a percentage (100 equals normal size, 200 equals double size, and so on). Read/write **Variant**.

*expression*.**Zoom**

*expression* Required. An expression that returns a **Window** object.

## Remarks

You can also set this property to **True** to make the window size fit the current selection.

This function affects only the sheet that's currently active in the window. To use this property on other sheets, you must first activate them.

## Example

[As it applies to the \*\*PageSetup\*\* object.](#)

This example scales Sheet1 by 150 percent when the worksheet is printed.

```
Worksheets("Sheet1").PageSetup.Zoom = 150
```



# ZOrder Property

Returns the z-order position of the object. Read-only **Long**.

## Remarks

In any collection of objects, the object at the back of the z-order is *collection*(1), and the object at the front of the z-order is *collection*(*collection.Count*). For example, if there are embedded charts on the active sheet, the chart at the back of the z-order is `ActiveSheet.ChartObjects(1)`, and the chart at the front of the z-order is `ActiveSheet.ChartObjects(ActiveSheet.ChartObjects.Count)`.

## Example

This example displays the z-order position of embedded chart one on Sheet1.

```
MsgBox "The chart's z-order position is " & _  
    Worksheets("Sheet1").ChartObjects(1).ZOrder
```



# Activate Event

Occurs when a workbook, worksheet, chart sheet, or embedded chart is activated.

**Private Sub** *object\_Activate()*

*object* **Chart**, **Workbook**, or **Worksheet**. For information about using events with the **Chart** object, see [Using Events with the Chart Object](#).

## Remarks

When you switch between two windows showing the same workbook, the `WindowActivate` event occurs, but the `Activate` event for the workbook doesn't occur.

This event doesn't occur when you create a new window.

## Example

This example sorts the range A1:A10 when the worksheet is activated.

```
Private Sub Worksheet_Activate()  
    Range("a1:a10").Sort Key1:=Range("a1"), Order:=xlAscending  
End Sub
```



# AddinInstall Event

Occurs when the workbook is installed as an add-in

**Private Sub Workbook\_AddinInstall()**

## Example

This example adds a control to the standard toolbar when the workbook is installed as an add-in.

```
Private Sub Workbook_AddinInstall()  
    With Application.Commandbars("Standard").Controls.Add  
        .Caption = "The AddIn's menu item"  
        .OnAction = "'ThisAddin.xls'!Amacro"  
    End With  
End Sub
```



# AddinUninstall Event

Occurs when the workbook is uninstalled as an add-in.

**Private Sub Workbook\_AddinUninstall()**

## **Remarks**

The add-in doesn't automatically close when it's uninstalled.

## Example

This example minimizes Microsoft Excel when the workbook is uninstalled as an add-in.

```
Private Sub Workbook_AddinUninstall()  
    Application.WindowState = xlMinimized  
End Sub
```



# AfterRefresh Event

Occurs after a query is completed or canceled.

**Private Sub QueryTable\_AfterRefresh(*Success As Boolean*)**

*Success* **True** if the query was completed successfully.

## Example

This example uses the Success argument to determine which section of code to run.

```
Private Sub QueryTable_AfterRefresh(Success As Boolean)
    If Success
        ' Query completed successfully
    Else
        ' Query failed or was cancelled
    End If
End Sub
```



[Show All](#)

# AfterXmlExport Event

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Occurs after Microsoft Excel saves or exports data from the specified workbook to an XML data file.

*expression.AfterXmlExport(Map, Url, Result)*

*expression* Required. An expression that returns a [Workbook](#) object.

**Map** Required [XmlMap](#) object. The schema map that was used to save or export data.

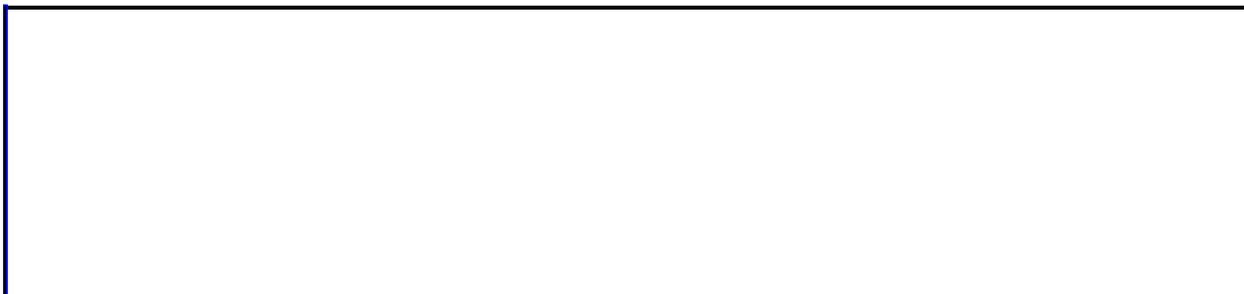
**Url** Required **String**. The location of the XML file that was exported.

**Result** Required [XlXmlExportResult](#). Indicates the results of the save or export operation.

**XlXmlExportResult** can be one of the following **XlXmlExportResult** constants:

**xlXmlExportSuccess** The XML data file was successfully exported.

**xlXmlExportValidationFailed** The contents of the XML data file do not match the specified schema map.



[Show All](#)

# AfterXmlImport Event

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Occurs after an existing XML data connection is refreshed or after new XML data is imported into the specified Microsoft Excel workbook.

*expression*.**AfterXmlImport**(*Map*, *IsRefresh*, *Result*)

*expression* Required. An expression that returns a [Workbook](#) object.

**Map** Required [XmlMap](#) object. The XML map that will be used to import data.

**IsRefresh** Required **Boolean**. **True** if the event was triggered by refreshing an existing connection to XML data; **False** if the event was triggered by importing from a different data source.

**Result** Required [IXmlImportResult](#). Indicates the results of the refresh or import operation.

**IXmlImportResult** can be one of the following **IXmlImportResult** constants:

**xlXmlImportElementsTruncated** The contents of the specified XML data file have been truncated because the XML data file is too large for the worksheet.

**xlXmlImportSuccess** The XML data file was successfully imported.

**xlXmlImportValidationFailed** The contents of the XML data file do not match the specified schema map.



# BeforeClose Event

Occurs before the workbook closes. If the workbook has been changed, this event occurs before the user is asked to save changes.

**Private Sub Workbook\_BeforeClose(*Cancel* As Boolean)**

*Cancel* **False** when the event occurs. If the event procedure sets this argument to **True**, the close operation stops and the workbook is left open.

## Example

This example always saves the workbook if it's been changed.

```
Private Sub Workbook_BeforeClose(Cancel as Boolean)
    If Me.Saved = False Then Me.Save
End Sub
```



[Show All](#)

# BeforeDoubleClick Event

 [Activate method as it applies to the \*\*Worksheet\*\* object.](#)

Occurs when a worksheet is double-clicked, before the default double-click action.

**Private Sub** *expression*\_BeforeDoubleClick(**ByVal** *Target* As Range, *Cancel* As Boolean)

*expression* A variable which references an object of type **Worksheet** declared with events in a class module.

**Target** Required. The cell nearest to the mouse pointer when the double-click occurs.

**Cancel** Optional. **False** when the event occurs. If the event procedure sets this argument to **True**, the default double-click action isn't performed when the procedure is finished.

 [Activate method as it applies to the \*\*Chart\*\* object.](#)

Occurs when an embedded chart is double-clicked, before the default double-click action.

**Private Sub** *expression*\_BeforeDoubleClick(**ByVal** *ElementID* As Long, **ByVal** *Arg1* As Long, **ByVal** *Arg2* As Long, *Cancel* As Boolean)

*expression* A variable which references an object of type **Chart** declared with events in a class module.

**Cancel** Optional. **False** when the event occurs. If the event procedure sets this argument to **True**, the default double-click action isn't performed when the procedure is finished.

**ElementID** Required. The double-clicked object The meaning of **Arg1** and **Arg2** depends on the **ElementID** value, as shown in the following table.

<i>ElementID</i>	<i>Arg1</i>	<i>Arg2</i>
<b>xlAxis</b>	AxisIndex	AxisType
<b>xlAxisTitle</b>	AxisIndex	AxisType
<b>xlDisplayUnitLabel</b>	AxisIndex	AxisType
<b>xlMajorGridlines</b>	AxisIndex	AxisType
<b>xlMinorGridlines</b>	AxisIndex	AxisType
<b>xlPivotChartDropZone</b>	DropZoneType	None
<b>xlPivotChartFieldButton</b>	DropZoneType	PivotFieldIndex
<b>xlDownBars</b>	GroupIndex	None
<b>xlDropLines</b>	GroupIndex	None
<b>xlHiLoLines</b>	GroupIndex	None
<b>xlRadarAxisLabels</b>	GroupIndex	None
<b>xlSeriesLines</b>	GroupIndex	None
<b>xlUpBars</b>	GroupIndex	None
<b>xlChartArea</b>	None	None
<b>xlChartTitle</b>	None	None
<b>xlCorners</b>	None	None
<b>xlDataTable</b>	None	None
<b>xlFloor</b>	None	None
<b>xlLegend</b>	None	None
<b>xlNothing</b>	None	None
<b>xlPlotArea</b>	None	None
<b>xlWalls</b>	None	None
<b>xlDataLabel</b>	SeriesIndex	PointIndex
<b>xlErrorBars</b>	SeriesIndex	None
<b>xlLegendEntry</b>	SeriesIndex	None
<b>xlLegendKey</b>	SeriesIndex	None
<b>xlSeries</b>	SeriesIndex	PointIndex
<b>xlTrendline</b>	SeriesIndex	TrendLineIndex
<b>xlXErrorBars</b>	SeriesIndex	None
<b>xlYErrorBars</b>	SeriesIndex	None
<b>xlShape</b>	ShapeIndex	None

The following table describes the meaning of the arguments.

<b>Argument</b>	<b>Description</b>
AxisIndex	Specifies whether the axis is primary or secondary. Can be one of the following <b>XIAxisGroup</b> constants: <b>xlPrimary</b> or <b>xlSecondary</b> .
AxisType	Specifies the axis type. Can be one of the following <b>XIAxisType</b> constants: <b>xlCategory</b> , <b>xlSeriesAxis</b> , or <b>xlValue</b> .
DropZoneType	Specifies the drop zone type: column, data, page, or row field. Can be one of the following <b>XIPivotFieldOrientation</b> constants: <b>xlColumnField</b> , <b>xlDataField</b> , <b>xlPageField</b> , or <b>xlRowField</b> . The column and row field constants specify the series and category fields, respectively.
GroupIndex	Specifies the offset within the <a href="#">ChartGroups</a> collection for a specific chart group.
PivotFieldIndex	Specifies the offset within the <a href="#">PivotFields</a> collection for a specific column (series), data, page, or row (category) field.
PointIndex	Specifies the offset within the <a href="#">Points</a> collection for a specific point within a series. The value – 1 indicates that all data points are selected.
SeriesIndex	Specifies the offset within the <a href="#">Series</a> collection for a specific series.
ShapeIndex	Specifies the offset within the <a href="#">Shapes</a> collection for a specific shape.
TrendlineIndex	Specifies the offset within the <a href="#">Trendlines</a> collection for a specific trendline within a series.

## Remarks

The [DoubleClick](#) method doesn't cause this event to occur.

This event doesn't occur when the user double-clicks the border of a cell.

## Example

 [As it applies to the \*\*Chart\*\* object.](#)

This example overrides the default double-click behavior for the chart floor.

```
Private Sub Chart_BeforeDoubleClick(ByVal ElementID As Long, _  
    ByVal Arg1 As Long, ByVal Arg2 As Long, Cancel As Boolean)  
  
    If ElementID = xlFloor Then  
        Cancel = True  
        MsgBox "Chart formatting for this item is restricted."  
    End If  
  
End Sub
```



# BeforePrint Event

Occurs before the workbook (or anything in it) is printed.

**Private Sub Workbook\_BeforePrint(*Cancel As Boolean*)**

*Cancel* **False** when the event occurs. If the event procedure sets this argument to **True**, the workbook isn't printed when the procedure is finished.

## Example

This example recalculates all worksheets in the active workbook before printing anything.

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
    For Each wk in Worksheets
        wk.Calculate
    Next
End Sub
```



# BeforeRefresh Event

Occurs before any refreshes of the query table. This includes refreshes resulting from calling the **Refresh** method, from the user's actions in the product, and from opening the workbook containing the query table.

**Private Sub QueryTable\_BeforeRefresh(*Cancel As Boolean*)**

*Cancel* **False** when the event occurs. If the event procedure sets this argument to **True**, the refresh doesn't occur when the procedure is finished.

## Example

This example runs before the query table is refreshed.

```
Private Sub QueryTable_BeforeRefresh(Cancel As Boolean)
    a = MsgBox("Refresh Now?", vbYesNoCancel)
    If a = vbNo Then Cancel = True
    MsgBox Cancel
End Sub
```



[Show All](#)

# BeforeRightClick Event

 [Activate method as it applies to the \*\*Worksheet\*\* object.](#)

Occurs when a worksheet is right-clicked, before the default right-click action.

**Private Sub** *expression*\_BeforeRightClick(**ByVal** *Target* As Range, *Cancel* As Boolean)

*expression* A variable which references an object of type **Worksheet** declared with events in a class module.

**Target** Required. The cell nearest to the mouse pointer when the right-click occurs.

**Cancel** Optional. **False** when the event occurs. If the event procedure sets this argument to **True**, the default right-click action doesn't occur when the procedure is finished.

 [Activate method as it applies to the \*\*Chart\*\* object.](#)

Occurs when an embedded chart is right-clicked, before the default right-click action.

**Private Sub** *expression*\_BeforeRightClick(*Cancel* As Boolean)

*expression* A variable which references an object of type **Chart** declared with events in a class module.

**Cancel** Required. **False** when the event occurs. If the event procedure sets this argument to **True**, the default right-click action isn't performed when the procedure is finished.

## Remarks

Like other worksheet events, this event doesn't occur if you right-click while the pointer is on a shape or a command bar (a toolbar or menu bar).

## Example

 [As it applies to the \*\*Worksheet\*\* object.](#)

This example adds a new menu item to the shortcut menu for cells B1:B10.

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, _
    Cancel As Boolean)
    Dim icbc As Object
    For Each icbc In Application.CommandBars("cell").Controls
        If icbc.Tag = "brccm" Then icbc.Delete
    Next icbc
    If Not Application.Intersect(Target, Range("b1:b10")) _
        Is Nothing Then
        With Application.CommandBars("cell").Controls _
            .Add(Type:=msoControlButton, before:=6, _
                temporary:=True)
            .Caption = "New Context Menu Item"
            .OnAction = "MyMacro"
            .Tag = "brccm"
        End With
    End If
End Sub
```



# BeforeSave Event

Occurs before the workbook is saved.

**Private Sub Workbook\_BeforeSave**(ByVal *SaveAsUi* As Boolean, *Cancel* As Boolean)

*SaveAsUi* **True** if the **Save As** dialog box will be displayed.

*Cancel* **False** when the event occurs. If the event procedure sets this argument to **True**, the workbook isn't saved when the procedure is finished.

## Example

This example prompts the user for a yes or no response before saving the workbook.

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, _  
    Cancel as Boolean)  
    a = MsgBox("Do you really want to save the workbook?", vbYesNo)  
    If a = vbNo Then Cancel = True  
End Sub
```



# BeforeXmlExport Event

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Occurs before Microsoft Excel saves or exports data from the specified workbook to an XML data file.

*expression*.**BeforeXmlExport**(*Map*, *Url*, *Cancel*)

*expression* Required. An expression that returns a **Workbook** object.

**Map** Required [XmlMap](#). The XML map that will be used to save or export data.

**Url** Required **String**. The location where you want to export the resulting XML file.

**Cancel** Required **Boolean**. Set to **True** to cancel the save or export operation.

## Remarks

This event will not occur when you are saving to the XML Spreadsheet file format.

--

# BeforeXmlImport Event

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Occurs before an existing XML data connection is refreshed or before new XML data is imported into a Microsoft Excel workbook.

*expression*.**BeforeXmlImport**(*Map*, *Url*, *IsRefresh*, *Cancel*)

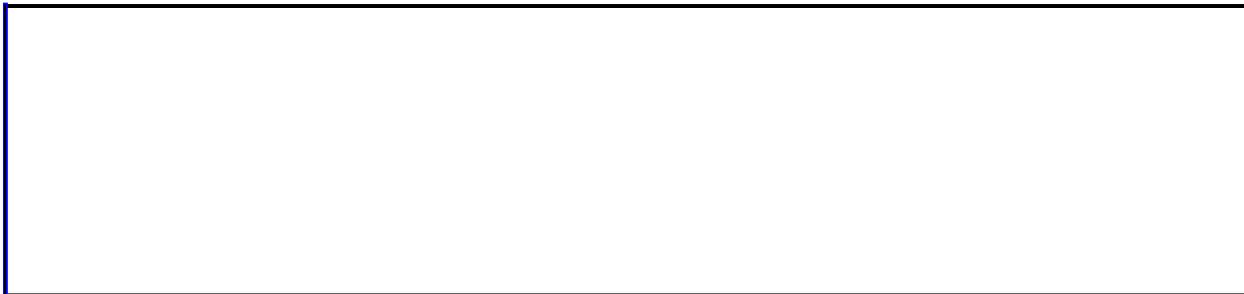
*expression* Required. An expression that returns one of the objects in the Applies To list.

**Map** Required [XmlMap](#) object. The XML map that will be used to import data.

**Url** Required **String**. The location of the XML file to be imported.

**IsRefresh** Required **Boolean**. **True** if the event was triggered by refreshing an existing connection to XML data; **False** if the event was triggered by importing from a different data source.

**Cancel** Required **Boolean**. Set to **True** to cancel the import or refresh operation.



# Calculate Event

Occurs after the chart plots new or changed data, for the **Chart** object. Occurs after the worksheet is recalculated, for the **Worksheet** object.

**Private Sub** *object*\_Calculate()

*object* **Chart** or **Worksheet**. For information about using events with the **Chart** object, see [Using Events with the Chart Object](#).

## Example

This example adjusts the size of columns A through F whenever the worksheet is recalculated.

```
Private Sub Worksheet_Calculate()  
    Columns("A:F").AutoFit  
End Sub
```



# Change Event

Occurs when cells on the worksheet are changed by the user or by an external link.

**Private Sub Worksheet\_Change(ByVal *Target* As Range)**

*Target* The changed range. Can be more than one cell.

## Remarks

This event doesn't occur when cells change during a recalculation. Use the Calculate event to trap a sheet recalculation.

## Example

This example changes the color of changed cells to blue.

```
Private Sub Worksheet_Change(ByVal Target as Range)
    Target.Font.ColorIndex = 5
End Sub
```



# Deactivate Event

Occurs when the chart, worksheet, or workbook is deactivated.

**Private Sub** *object\_Deactivate()*

*object* **Chart, Workbook, or Worksheet.** For information about using events with the **Chart** object, see [Using Events with the Chart Object](#).

## Example

This example arranges all open windows when the workbook is deactivated.

```
Private Sub Workbook_Deactivate()  
    Application.Windows.Arrange xlArrangeStyleTiled  
End Sub
```



# DragOver Event

Occurs when a range of cells is dragged over a chart.

**Private Sub** *object*\_DragOver()

*object* An object of type **Chart** declared with events in a class module. For more information, see [Using Events with the Chart Object](#).

## Example

This example displays the address of a range of cells dragged over a chart.

```
Private Sub Chart_DragOver()  
    MsgBox Selection.Address  
End Sub
```



# DragPlot Event

Occurs when a range of cells is dragged and dropped on a chart.

**Private Sub** *object*\_DragPlot()

*object* An object of type **Chart** declared with events in a class module. For more information, see [Using Events with the Chart Object](#).

## Example

This example changes the chart type when a range of cells is dragged and dropped on a chart.

```
Private Sub Chart_DragPlot()  
    Me.ChartType = xlLine  
End Sub
```



# FollowHyperlink Event

Occurs when you click any hyperlink on a worksheet. For application- and workbook-level events, see the [SheetFollowHyperlink](#) event.

**Private Sub Worksheet\_FollowHyperlink(ByVal *Target* As Hyperlink)**

*Target* Required **Hyperlink**. A **Hyperlink** object that represents the destination of the hyperlink.

## Example

This example keeps a list, or history, of all the links that have been visited from the active worksheet.

```
Private Sub Worksheet_FollowHyperlink(ByVal Target As Hyperlink)
    With UserForm1
        .ListBox1.AddItem Target.Address
        .Show
    End With
End Sub
```



# GotFocus Event

Occurs when an ActiveX control gets input focus.

**Private Sub *object*\_GotFocus()**

*object* The name of an ActiveX control.

## Example

This example runs when ListBox1 gets the focus.

```
Private Sub ListBox1_GotFocus()  
    ' runs when list box gets the focus  
End Sub
```



# LostFocus Event

Occurs when an ActiveX control loses input focus.

**Private Sub *object*\_LostFocus()**

*object* The name of an ActiveX control.

## Example

This example runs when ListBox1 loses the focus.

```
Private Sub ListBox1_LostFocus()  
    ' runs when list box loses the focus  
End Sub
```



# MouseDown Event

Occurs when a mouse button is pressed while the pointer is over a chart.

**Private Sub *object*\_MouseDown(ByVal *Button* As Long, ByVal *Shift* As Long, ByVal *X* As Long, ByVal *Y* As Long)**

*object* An object of type **Chart** declared with events in a class module. For more information, see [Using Events with the Chart Object](#).

*Button* The mouse button that was pressed. Can be one of the following **XlMouseButton** constants: **xlNoButton**, **xlPrimaryButton**, **xlSecondaryButton**, or **xlMiddleButton**.

*Shift* The state of the SHIFT, CTRL, and ALT keys when the event occurred. Can be one of or a sum of the following values.

Value	Meaning
-------	---------

0 (zero)	No keys
----------	---------

1	SHIFT key
---	-----------

2	CTRL key
---	----------

4	ALT key
---	---------

<i>X</i>	The X coordinate of the mouse pointer in chart object client coordinates.
----------	---

<i>Y</i>	The Y coordinate of the mouse pointer in chart object client coordinates.
----------	---

## Example

This example runs when a mouse button is pressed while the pointer is over a chart.

```
Private Sub Chart_MouseDown(ByVal Button As Long, _  
    ByVal Shift As Long, ByVal X As Long, ByVal Y As Long)  
    MsgBox "Button = " & Button & chr$(13) & _  
        "Shift = " & Shift & chr$(13) & _  
        "X = " & X & " Y = " & Y  
End Sub
```



# MouseMove Event

Occurs when the position of the mouse pointer changes over a chart.

**Private Sub *object*\_MouseMove(ByVal *Button* As Long, ByVal *Shift* As Long, ByVal *X* As Long, ByVal *Y* As Long)**

*object* An object of type **Chart** declared with events in a class module. For more information, see [Using Events with the Chart Object](#).

*Button* The mouse button that was pressed. Can be one of the following **XlMouseButton** constants: **xlNoButton**, **xlPrimaryButton**, **xlSecondaryButton**, or **xlMiddleButton**.

*Shift* The state of the SHIFT, CTRL, and ALT keys when the event occurred. Can be one of or a sum of the following values.

Value	Meaning
-------	---------

0 (zero)	No keys
----------	---------

1	SHIFT key
---	-----------

2	CTRL key
---	----------

4	ALT key
---	---------

<i>X</i>	The X coordinate of the mouse pointer in chart object client coordinates.
----------	---

<i>Y</i>	The Y coordinate of the mouse pointer in chart object client coordinates.
----------	---

## Example

This example runs when the position of the mouse pointer changes over a chart.

```
Private Sub Chart_MouseMove(ByVal Button As Long, ByVal Shift As Long, ByVal X As Single, ByVal Y As Single)
    MsgBox "X = " & X & " Y = " & Y
End Sub
```



# MouseUp Event

Occurs when a mouse button is released while the pointer is over a chart.

**Private Sub *object*\_MouseDown(ByVal *Button* As Long, ByVal *Shift* As Long, ByVal *X* As Long, ByVal *Y* As Long)**

*object* An object of type **Chart** declared with events in a class module. For more information, see [Using Events with the Chart Object](#).

*Button* The mouse button that was released. Can be one of the following **XlMouseButton** constants: **xlNoButton**, **xlPrimaryButton**, **xlSecondaryButton**, or **xlMiddleButton**.

*Shift* The state of the SHIFT, CTRL, and ALT keys when the event occurred. Can be one of or a sum of the following values.

Value	Meaning
-------	---------

0 (zero)	No keys
----------	---------

1	SHIFT key
---	-----------

2	CTRL key
---	----------

4	ALT key
---	---------

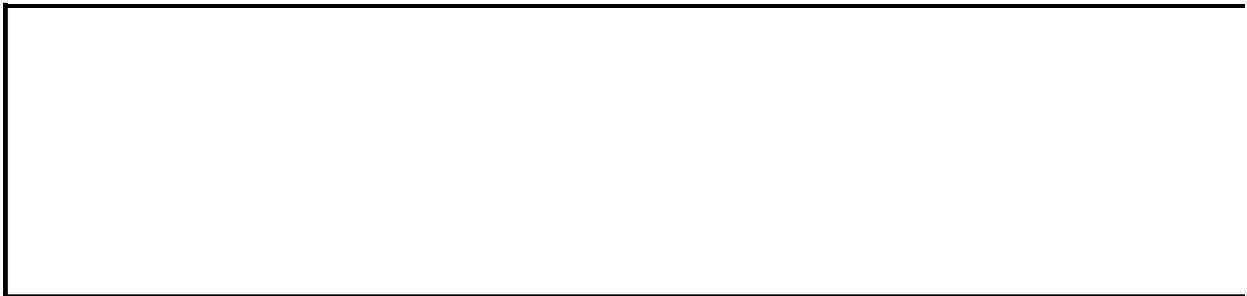
*X* The X coordinate of the mouse pointer in chart object client coordinates.

*Y* The Y coordinate of the mouse pointer in chart object client coordinates.

## Example

This example runs when a mouse button is released over a chart.

```
Private Sub Chart_MouseUp(ByVal Button As Long, _  
    ByVal Shift As Long, ByVal X As Long, ByVal Y As Long)  
    MsgBox "Button = " & Button & chr$(13) & _  
        "Shift = " & Shift & chr$(13) & _  
        "X = " & X & " Y = " & Y  
End Sub
```



# NewSheet Event

Occurs when a new sheet is created in the workbook.

**Private Sub Workbook\_NewSheet(ByVal *Sh* As Object)**

*Sh* The new sheet. Can be a **Worksheet** or **Chart** object.

## Example

This example moves new sheets to the end of the workbook.

```
Private Sub Workbook_NewSheet(ByVal Sh as Object)  
    Sh.Move After:= Sheets(Sheets.Count)  
End Sub
```



# NewWorkbook Event

Occurs when a new workbook is created.

**Private Sub *object*\_NewWorkbook(ByVal *Wb* As Workbook)**

*object* An object of type **Application** declared with events in a class module.  
For more information, see [Using Events with the Application Object](#).

*Wb* The new workbook.

## Example

This example arranges open windows when a new workbook is created.

```
Private Sub App_NewWorkbook(ByVal Wb As Workbook)
    Application.Windows.Arrange xlArrangeStyleTiled
End Sub
```



# Open Event

Occurs when the workbook is opened.

**Private Sub Workbook\_Open()**

## Example

This example maximizes Microsoft Excel whenever the workbook is opened.

```
Private Sub Workbook_Open()  
    Application.WindowState = xlMaximized  
End Sub
```



# PivotTableCloseConnection Event

Occurs after a PivotTable report closes the connection to its data source.

**Private Sub** *expression*\_PivotTableCloseConnection(**ByVal** *Target* As PivotTable)

*expression* A variable which references an object of type **Workbook** declared with events in a class module.

**Target** Required. The selected PivotTable report.

## Example

This example displays a message stating that the PivotTable report's connection to its source has been closed. This example assumes you have declared an object of type **Workbook** with events in a class module.

```
Private Sub ConnectionApp_PivotTableCloseConnection(ByVal Target As  
    MsgBox "The PivotTable connection has been closed."  
End Sub
```



# PivotTableOpenConnection Event

Occurs after a PivotTable report opens the connection to its data source.

**Private Sub** *expression*\_PivotTableOpenConnection(**ByVal** *Target* As PivotTable)

*expression* A variable which references an object of type **Workbook** declared with events in a class module.

**Target** Required. The selected PivotTable report.

## Example

This example displays a message stating that the PivotTable report's connection to its source has been opened. This example assumes you have declared an object of type **Workbook** with events in a class module.

```
Private Sub ConnectionApp_PivotTableOpenConnection(ByVal Target As P  
    MsgBox "The PivotTable connection has been opened."  
End Sub
```



# PivotTableUpdate Event

Occurs after a PivotTable report is updated on a worksheet.

**Private Sub** *expression*\_PivotTableUpdate(**ByVal** *Target* **As** PivotTable)

*expression* A variable which references an object of type **Worksheet** declared with events in a class module.

**Target** Required. The selected PivotTable report.

## Example

This example displays a message stating that the PivotTable report has been updated. This example assumes you have declared an object of type **Worksheet** with events in a class module.

```
Private Sub Worksheet_PivotTableUpdate(ByVal Target As PivotTable)
    MsgBox "The PivotTable connection has been updated."
End Sub
```



# Resize Event

Occurs when the chart is resized.

**Private Sub** *object*\_Resize()

*object* **Chart** or an object of type **Chart** declared with events in a class module. For more information, see [Using Events with Embedded Charts](#).

## Example

This example keeps the upper-left corner of the chart at the same location when the chart is resized.

```
Private Sub myChartClass_Resize()  
    With ActiveChart.Parent  
        .Left = 100  
        .Top = 150  
    End With  
End Sub
```



# Select Event

Occurs when a chart element is selected.

**Private Sub *object\_Select*(ByVal *ElementID* As Long, ByVal *Arg1* As Long, ByVal *Arg2* As Long)**

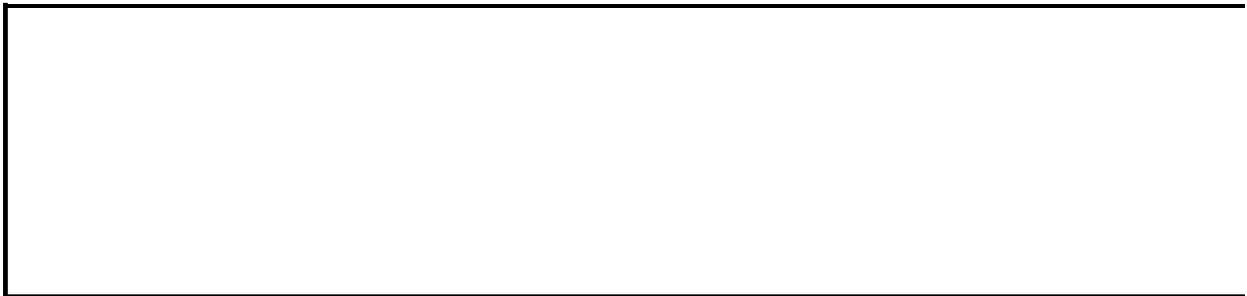
*object* **Chart** or an object of type **Chart** declared with events in a class module. For more information, see [Using Events with Embedded Charts](#).

*ElementID* , *Arg1* , *Arg2* The selected chart element. For more information about these arguments, see the [BeforeDoubleClick](#) event.

## Example

This example displays a message box if the user selects the chart title.

```
Private Sub Chart_Select(ByVal ElementID As Long, _  
    ByVal Arg1 As Long, ByVal Arg2 As Long)  
    If ElementId = xlChartTitle Then  
        MsgBox "please don't change the chart title"  
    End If  
End Sub
```



# SelectionChange Event

Occurs when the selection changes on a worksheet.

**Private Sub Worksheet\_SelectionChange(ByVal *Target* As Excel.Range)**

*Target* The new selected range.

## Example

This example scrolls through the workbook window until the selection is in the upper-left corner of the window.

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    With ActiveWindow
        .ScrollRow = Target.Row
        .ScrollColumn = Target.Column
    End With
End Sub
```



# SeriesChange Event

Occurs when the user changes the value of a chart data point.

**Private Sub *object\_SeriesChange*(ByVal *SeriesIndex* As Long, ByVal *PointIndex* As Long)**

*object* An object of type **Chart** declared with events in a class module. For more information, see [Using Events with the Chart Object](#).

*SeriesIndex* The offset within the **Series** collection for the changed series.

*PointIndex* The offset within the **Points** collection for the changed point.

## Example

This example changes the point's border color when the user changes the point value.

```
Private Sub Chart_SeriesChange(ByVal SeriesIndex As Long, _  
    ByVal PointIndex As Long)  
    Set p = Me.SeriesCollection(SeriesIndex).Points(PointIndex)  
    p.Border.ColorIndex = 3  
End Sub
```



# SheetActivate Event

Occurs when any sheet is activated.

**Private Sub *object*\_SheetActivate(ByVal *Sh* As Object)**

*object* **Application** or **Workbook**.

*Sh* The activated sheet. Can be a **Chart** or **Worksheet** object.

## Example

This example displays the name of each activated sheet.

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
    MsgBox Sh.Name
End Sub
```



# SheetBeforeDoubleClick Event

Occurs when any worksheet is double-clicked, before the default double-click action.

**Private Sub** *object\_SheetBeforeDoubleClick*(**ByVal** *Sh* **As Object**, **ByVal** *Target* **As Range**, **ByVal** *Cancel* **As Boolean**)

*object* **Application** or **Workbook**. For more information about using events with the **Application** object, see [Using Events with the Application Object](#).

*Sh* A **Worksheet** object that represents the sheet.

*Target* The cell nearest to the mouse pointer when the double-click occurred.

*Cancel* **False** when the event occurs. If the event procedure sets this argument to **True**, the default double-click action isn't performed when the procedure is finished.

## Remarks

This event doesn't occur on chart sheets.

## Example

This example disables the default double-click action.

```
Private Sub Workbook_SheetBeforeDoubleClick(ByVal Sh As Object, _  
    ByVal Target As Range, ByVal Cancel As Boolean)  
    Cancel = True  
End Sub
```



# SheetBeforeRightClick Event

Occurs when any worksheet is right-clicked, before the default right-click action.

**Private Sub** *object\_SheetBeforeRightClick*(**ByVal** *Sh* As **Object**, **ByVal** *Target* As **Range**, **ByVal** *Cancel* As **Boolean**)

*object* **Application** or **Workbook**. For more information about using events with the **Application** object, see [Using Events with the Application Object](#).

*Sh* A **Worksheet** object that represents the sheet.

*Target* The cell nearest to the mouse pointer when the right-click occurred.

*Cancel* **False** when the event occurs. If the event procedure sets this argument to **True**, the default right-click action isn't performed when the procedure is finished.

## Remarks

This event doesn't occur on chart sheets.

## Example

This example disables the default right-click action. For another example, see the [BeforeRightClick](#) event example.

```
Private Sub Workbook_SheetBeforeRightClick(ByVal Sh As Object, _  
    ByVal Target As Range, ByVal Cancel As Boolean)  
    Cancel = True  
End Sub
```



# SheetCalculate Event

Occurs after any worksheet is recalculated or after any changed data is plotted on a chart.

**Private Sub** *object\_SheetCalculate*(ByVal *Sh* As Object)

*object* **Application** or **Workbook**. For more information about using events with the **Application** object, see [Using Events with the Application Object](#).

*Sh* The sheet. Can be a **Chart** or **Worksheet** object.

## Example

This example sorts the range A1:A100 on worksheet one when any sheet in the workbook is calculated.

```
Private Sub Workbook_SheetCalculate(ByVal Sh As Object)
    With Worksheets(1)
        .Range("a1:a100").Sort Key1:=.Range("a1")
    End With
End Sub
```



# SheetChange Event

Occurs when cells in any worksheet are changed by the user or by an external link.

**Private Sub *object*\_SheetChange(ByVal *Sh* As Object, ByVal *Source* As Range)**

*object* **Application** or **Workbook**. For more information about using events with the **Application** object, see [Using Events with the Application Object](#).

*Sh* A **Worksheet** object that represents the sheet.

*Source* The changed range.

## Remarks

This event doesn't occur on chart sheets.

## Example

This example runs when any worksheet is changed.

```
Private Sub Workbook_SheetChange(ByVal Sh As Object, _  
    ByVal Source As Range)  
    ' runs when a sheet is changed  
End Sub
```



# SheetDeactivate Event

Occurs when any sheet is deactivated.

**Private Sub** *object\_SheetDeactivate*(ByVal *Sh* As Object)

*object* **Application** or **Workbook**.

*Sh* The sheet. Can be a **Chart** or **Worksheet** object.

## Example

This example displays the name of each deactivated sheet.

```
Private Sub Workbook_SheetDeactivate(ByVal Sh As Object)
    MsgBox Sh.Name
End Sub
```



# SheetFollowHyperlink Event

Occurs when you click any hyperlink in Microsoft Excel. For worksheet-level events, see the Help topic for the [FollowHyperlink](#) event.

**Private Sub Workbook\_SheetFollowHyperlink(ByVal *Sh* As Object, ByVal *Target* As Hyperlink)**

*Sh* Required **Object**. The **Worksheet** object that contains the hyperlink.

*Target* Required **Hyperlink**. The **Hyperlink** object that represents the destination of the hyperlink.

## Example

This example keeps a list, or history, of all the hyperlinks in the current workbook that have been clicked, plus the names of the worksheets that contain these hyperlinks.

```
Private Sub Workbook_SheetFollowHyperlink(ByVal Sh as Object, _  
    ByVal Target As Hyperlink)  
    UserForm1.ListBox1.AddItem Sh.Name & ":" & Target.Address  
    UserForm1.Show  
End Sub
```



# SheetPivotTableUpdate Event

Occurs after the sheet of the PivotTable report has been updated.

**Private Sub** *expression\_SheetPivotTableUpdate*(ByVal *Sh* As Object, *Target* As PivotTable)

*expression* A variable which references an object of type **Application** or **Workbook** declared with events in a class module.

*Sh* Required. The selected sheet.

*Target* Required. The selected PivotTable report.

## Example

This example displays a message stating that the sheet of the PivotTable report has been updated. This example assumes you have declared an object of type **Application** or **Workbook** with events in a class module.

```
Private Sub ConnectionApp_SheetPivotTableUpdate(ByVal shOne As Objec  
    MsgBox "The SheetPivotTable connection has been updated."  
End Sub
```



# SheetSelectionChange Event

Occurs when the selection changes on any worksheet (doesn't occur if the selection is on a chart sheet).

**Private Sub *object*\_SheetSelectionChange(ByVal *Sh* As Object, ByVal *Target* As Excel.Range)**

*object* **Application** or **Workbook**. For more information about using events with the **Application** object, see [Using Events with the Application Object](#).

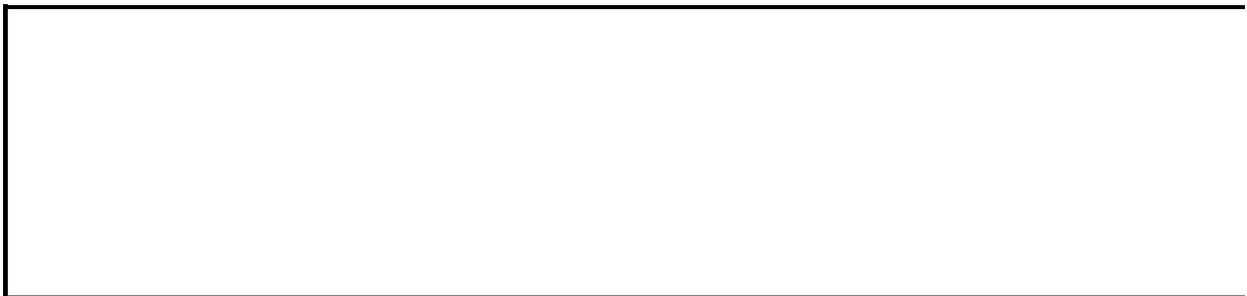
*Sh* The worksheet that contains the new selection.

*Target* The new selected range.

## Example

This example displays the sheet name and address of the selected range in the status bar.

```
Private Sub Workbook_SheetSelectionChange(ByVal Sh As Object, _  
    ByVal Target As Excel.Range)  
    Application.StatusBar = Sh.Name & ":" & Target.Address  
End Sub
```



[Show All](#)

# Sync Event

Occurs when the local copy of a worksheet that is part of a Document Workspace is synchronized with the copy on the server.

**Private Sub** *object\_Sync(SyncEventType)*

*object* An object of type **Workbook** declared by using the **WithEvents** keyword in a [class module](#).

**SyncEventType** **MsoSyncEventType**. The status of the worksheet synchronization.

**MsoSyncEventType** can be one of the following **msoSyncEventType** constants:

**msoSyncEventDownloadFailed**

**msoSyncEventDownloadInitiated**

**msoSyncEventDownloadNoChange**

**msoSyncEventDownloadSucceeded**

**msoSyncEventOffline**

**msoSyncEventUploadFailed**

**msoSyncEventUploadInitiated**

**msoSyncEventUploadSucceeded**

## Example

The following example displays a message if the synchronization of a worksheet in a Document Workspace fails.

```
Private Sub Worksheet_Sync(ByVal SyncEventType As Office.MsoSyncEven
    If SyncEventType = msoSyncEventDownloadFailed Or _
        SyncEventType = msoSyncEventUploadFailed Then
        MsgBox "Document synchronization failed. " & _
            "Please contact your administrator " & vbCrLf & _
            "or try again later."
    End If
End Sub
```



# WindowActivate Event

Occurs when any workbook window is activated.

**Private Sub *object*\_WindowActivate(ByVal *Wb* As Excel.Workbook, ByVal *Wn* As Excel.Window)**

*object* **Application** or **Workbook**. For more information about using events with the **Application** object, see [Using Events with the Application Object](#).

*Wb* Used only with the **Application** object. The workbook displayed in the activated window.

*Wn* The activated window.

## Example

This example maximizes any workbook window when it's activated.

```
Private Sub Workbook_WindowActivate(ByVal Wn As Excel.Window)
    Wn.WindowState = xlMaximized
End Sub
```



# WindowDeactivate Event

Occurs when any workbook window is deactivated.

**Private Sub** *object*\_WindowDeactivate(**ByVal** *Wb* **As** Excel.Workbook,  
**ByVal** *Wn* **As** Excel.Window)

*object* **Application** or **Workbook**. For more information about using events with the **Application** object, see [Using Events with the Application Object](#).

*Wb* Used only with the **Application** object. The workbook displayed in the deactivated window.

*Wn* The deactivated window.

## Example

This example minimizes any workbook window when it's deactivated.

```
Private Sub Workbook_WindowDeactivate(ByVal Wn As Excel.Window)
    Wn.WindowState = xlMinimized
End Sub
```



# WindowResize Event

Occurs when any workbook window is resized.

**Private Sub *object*\_WindowResize(ByVal *Wb* As Excel.Workbook, ByVal *Wn* As Excel.Window)**

*object* **Application** or **Workbook**. For more information about using events with the **Application** object, see [Using Events with the Application Object](#).

*Wb* Used only with the **Application** object. The workbook displayed in the resized window.

*Wn* The resized window.

## Example

This example runs when any workbook window is resized.

```
Private Sub Workbook_WindowResize(ByVal Wn As Excel.Window)
    Application.StatusBar = Wn.Caption & " resized"
End Sub
```



# WorkbookActivate Event

Occurs when any workbook is activated.

**Private Sub *app*\_WorkbookActivate(ByVal *Wb* As Workbook)**

*app* An object of type **Application** declared with events in a class module. For more information, see [Using Events with the Application Object](#).

*Wb* The activated workbook.

## Example

This example arranges open windows when a workbook is activated.

```
Private Sub App_WorkbookActivate(ByVal Wb As Workbook)
    Application.Windows.Arrange xlArrangeStyleTiled
End Sub
```



# WorkbookAddinInstall Event

Occurs when a workbook is installed as an add-in.

**Private Sub *object*\_WorkbookAddinInstall(ByVal *Wb* As Workbook)**

*object* An object of type **Application** declared with events in a class module.  
For more information, see [Using Events with the Application Object](#).

*Wb* The installed workbook.

## Example

This example maximizes the Microsoft Excel window when a workbook is installed as an add-in.

```
Private Sub App_WorkbookAddinInstall(ByVal Wb As Workbook)
    Application.WindowState = xlMaximized
End Sub
```



# WorkbookAddinUninstall Event

Occurs when any add-in workbook is uninstalled.

**Private Sub *object*\_WorkbookAddinUninstall(ByVal *Wb* As Workbook)**

*object* An object of type **Application** declared with events in a class module.  
For more information, see [Using Events with the Application Object](#).

*Wb* The uninstalled workbook.

## Example

This example minimizes the Microsoft Excel window when a workbook is installed as an add-in.

```
Private Sub App_WorkbookAddinUninstall(ByVal Wb As Workbook)
    Application.WindowState = xlMinimized
End Sub
```



[Show All](#)

# WorkbookAfterXmlExport Event

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Occurs after Microsoft Excel saves or exports data from any open workbook to an XML data file.

*expression*. **WorkbookAfterXmlExport(Wb, Map, Url, Result)**

*expression* Required. An expression that returns an **Application** object.

**Wb** Required **Workbook**. The target workbook.

**Map** Required **XmlMap**. The XML map that was used to save or export data.

**Url** Required **String**. The location of the XML file that was exported.

**Result** Required [XlXmlExportResult](#). Indicates the results of the save or export operation.

XlXmlExportResult can be one of the following XlXmlExportResult constants

**xlXmlExportSuccess** The XML data file was successfully exported.

**xlXmlExportValidationFailed** The contents of the XML data file do not match the specified schema map.

## Remarks

Use the **AfterXmlExport** event if you want to perform an operation after XML data has been exported from a particular workbook.

[Show All](#)

# WorkbookAfterXmlImport Event

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Occurs after an existing XML data connection is refreshed, or new XML data is imported into any open Microsoft Excel workbook.

*expression*. **WorkbookAfterXmlImport(Wb, Map, IsRefresh, Result)**

*expression* Required. An expression that returns an **Application** object.

**Wb** Required **Workbook**. The target workbook.

**Map** Required **XmlMap**. The XML map that was used to import data.

**IsRefresh** Required **Boolean**. **True** if the event was triggered by refreshing an existing connection to XML data, **False** if a new mapping was created.

**Result** Required [xlXmlImportResult](#). Indicates the results of the refresh or import operation.

xlXmlImportResult can be one of the following xlXmlImportResult constants

**xlXmlImportElementsTruncated** The contents of the specified XML data file have been truncated because the XML data file is too large for the worksheet.

**xlXmlImportSuccess** The XML data file was successfully imported.

**xlXmlImportValidationFailed** The contents of the XML data file do not match the specified schema map.

## Remarks

Use the **AfterXmlImport** event if you want to perform an operation after XML data has been imported into a particular workbook.

--

# WorkbookBeforeClose Event

Occurs immediately before any open workbook closes.

**Private Sub *object*\_WorkbookBeforeClose(ByVal *Wb* As Workbook, ByVal *Cancel* As Boolean)**

*object* An object of type **Application** declared with events in a class module. For more information, see [Using Events with the Application Object](#).

*Wb* The workbook that's being closed.

*Cancel* **False** when the event occurs. If the event procedure sets this argument to **True**, the workbook doesn't close when the procedure is finished.

## Example

This example prompts the user for a yes or no response before closing any workbook.

```
Private Sub App_WorkbookBeforeClose(ByVal Wb as Workbook, _  
    Cancel as Boolean)  
    a = MsgBox("Do you really want to close the workbook?", _  
        vbYesNo)  
    If a = vbNo Then Cancel = True  
End Sub
```



# WorkbookBeforePrint Event

Occurs before any open workbook is printed.

**Private Sub *object*\_WorkbookBeforePrint(ByVal *Wb* As Workbook, ByVal *Cancel* As Boolean)**

*object* An object of type **Application** declared with events in a class module. For more information, see [Using Events with the Application Object](#).

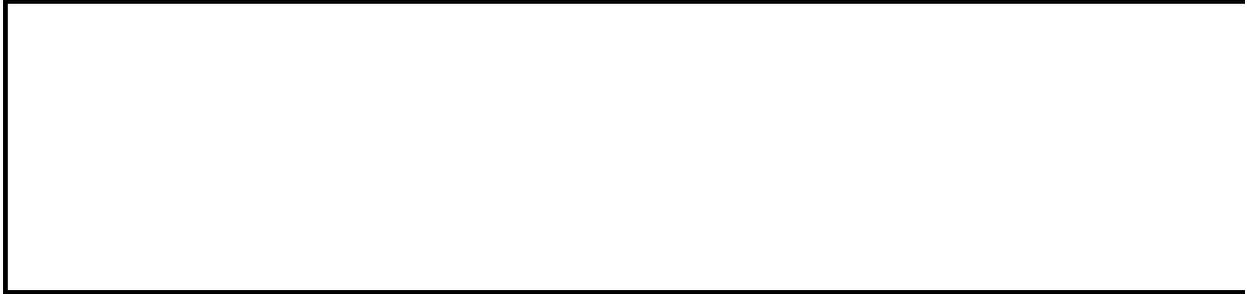
*Wb* The workbook.

*Cancel* **False** when the event occurs. If the event procedure sets this argument to **True**, the workbook isn't printed when the procedure is finished.

## Example

This example recalculates all worksheets in the workbook before printing anything.

```
Private Sub App_WorkbookBeforePrint(ByVal Wb As Workbook, _  
    Cancel As Boolean)  
    For Each wk in Wb.Worksheets  
        wk.Calculate  
    Next  
End Sub
```



# WorkbookBeforeSave Event

Occurs before any open workbook is saved.

**Private Sub *object*\_WorkbookBeforeSave(ByVal Wb As Workbook, ByVal SaveAsUi As Boolean, ByVal Cancel As Boolean)**

*object* An object of type **Application** declared with events in a class module. For more information, see [Using Events with the Application Object](#).

*Wb* The workbook.

*SaveAsUi* **True** if the **Save As** dialog box will be displayed.

*Cancel* **False** when the event occurs. If the event procedure sets this argument to **True**, the workbook isn't saved when the procedure is finished.

## Example

This example prompts the user for a yes or no response before saving any workbook.

```
Private Sub App_WorkbookBeforeSave(ByVal Wb As Workbook, _  
    ByVal SaveAsUI As Boolean, Cancel as Boolean)  
    a = MsgBox("Do you really want to save the workbook?", vbYesNo)  
    If a = vbNo Then Cancel = True  
End Sub
```



# WorkbookBeforeXmlExport Event

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Occurs before Microsoft Excel saves or exports data from any open workbook to an XML data file.

*expression*. **WorkbookBeforeXmlExport**(*Wb*, *Map*, *Url*, *Cancel*)

*expression* Required. An expression that returns an **Application** object.

**Wb** Required **Workbook**. The target workbook.

**Map** Required **XmlMap**. The XML map that will be used to save or export data.

**Url** Required **String**. The location of the XML file to be exported.

**Cancel** Required **Boolean**. Set to **True** to cancel the save or export operation.

## Remarks

Use the **BeforeXmlExport** event if you want to capture XML data that is being exported or saved from a particular workbook.

--

# WorkbookBeforeXmlImport Event

**Note** XML features, except for saving files in the XML Spreadsheet format, are available only in Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003.

Occurs before an existing XML data connection is refreshed, or new XML data is imported into any open Microsoft Excel workbook.

*expression*. **WorkbookBeforeXmlImport**(*Wb*, *Map*, *Url*, *IsRefresh*, *Cancel*)

*expression* Required. An expression that returns an **Application** object.

**Wb** Required **Workbook**. The target workbook.

**Map** Required **XmlMap**. The XML map that will be used to import data.

**Url** Required **String**. The location of the XML file to be imported.

**IsRefresh** Required **Boolean**. **True** if the event was triggered by refreshing an existing connection to XML data, **False** if a new mapping will be created.

**Cancel** Required **Boolean**. **Cancel** Required **Boolean**. Set to **True** to cancel the import or refresh operation.

## Remarks

Use the **BeforeXmlImport** event if you want to capture XML data that is being imported or refreshed to a particular workbook.

--

# WorkbookDeactivate Event

Occurs when any open workbook is deactivated.

**Private Sub *object*\_WorkbookDeactivate(ByVal *Wb* As Workbook)**

*object* An object of type **Application** declared with events in a class module.  
For more information, see [Using Events with the Application Object](#).

*Wb* The workbook.

## Example

This example arranges all open windows when a workbook is deactivated.

```
Private Sub App_WorkbookDeactivate(ByVal Wb As Workbook)
    Application.Windows.Arrange xlArrangeStyleTiled
End Sub
```



# WorkbookNewSheet Event

Occurs when a new sheet is created in any open workbook.

**Private Sub *object*\_WorkbookNewSheet(ByVal *Wb* As Workbook, ByVal *Sh* As Object)**

*object* An object of type **Application** declared with events in a class module. For more information, see [Using Events with the Application Object](#).

*Wb* The workbook.

*Sh* The new sheet.

## Example

This example moves the new sheet to the end of the workbook.

```
Private Sub App_WorkbookNewSheet(ByVal Wb As Workbook, _  
    ByVal Sh As Object)  
    Sh.Move After:=Wb.Sheets(Wb.Sheets.Count)  
End Sub
```



# WorkbookOpen Event

Occurs when a workbook is opened.

**Private Sub *object*\_WorkbookOpen(ByVal *Wb* As Workbook)**

*object* An object of type **Application** declared with events in a class module.  
For more information, see [Using Events with the Application Object](#).

*Wb* The workbook.

## Example

This example arranges all open windows when a workbook is opened.

```
Private Sub App_WorkbookOpen(ByVal Wb As Workbook)
    Application.Windows.Arrange xlArrangeStyleTiled
End Sub
```



# WorkbookPivotTableCloseConnection Event

Occurs after a PivotTable report connection has been closed.

**Private Sub** *expression*\_WorkbookPivotTableCloseConnection(**ByVal** *Wb* As Workbook, *Target* As PivotTable)

*expression* A variable which references an object of type **Application** declared with events in a class module.

**Wb** Required. The selected workbook.

**Target** Required. The selected PivotTable report.

## Example

This example displays a message stating that the PivotTable report's connection to its source has been closed. This example assumes you have declared an object of type **Workbook** with events in a class module.

```
Private Sub ConnectionApp_WorkbookPivotTableCloseConnection(ByVal wb
    MsgBox "The PivotTable connection has been closed."
End Sub
```



# WorkbookPivotTableOpenConnection Event

Occurs after a PivotTable report connection has been opened.

**Private Sub** *expression*\_WorkbookPivotTableOpenConnection(**ByVal** *Wb* As Workbook, *Target* As PivotTable)

*expression* A variable which references an object of type **Application** declared with events in a class module.

**Wb** Required. The selected workbook.

**Target** Required. The selected PivotTable report.

## Example

This example displays a message stating that the PivotTable report's connection to its source has been opened. This example assumes you have declared an object of type **Workbook** with events in a class module.

```
Private Sub ConnectionApp_WorkbookPivotTableOpenConnection(ByVal wbC  
    MsgBox "The PivotTable connection has been opened."  
End Sub
```



[Show All](#)

# WorkbookSync Event

Occurs when the local copy of a workbook that is part of a Document Workspace is synchronized with the copy on the server.

**Private Sub** *object\_WorkbookSync*(*Wb*, *SyncEventType*)

*object* An object of type **Workbook** declared by using the **WithEvents** keyword in a [class module](#).

**Wb** **Workbook**. The workbook being synchronized.

**SyncEventType** Required [MsoSyncEventType](#). The status of the workbook synchronization.

**MsoSyncEventType** can be one of the following **msoSyncEventType** constants:

**msoSyncEventDownloadFailed**

**msoSyncEventDownloadInitiated**

**msoSyncEventDownloadNoChange**

**msoSyncEventDownloadSucceeded**

**msoSyncEventOffline**

**msoSyncEventUploadFailed**

**msoSyncEventUploadInitiated**

**msoSyncEventUploadSucceeded**

## Example

The following example displays a message if the synchronization of a workbook in a Document Workspace fails.

```
Private Sub app_WorkbookSync(ByVal Wb As Workbook, _  
    ByVal SyncEventType As Office.MsoSyncEventType)  
  
    If SyncEventType = msoSyncEventDownloadFailed Or _  
        SyncEventType = msoSyncEventUploadFailed Then  
  
        MsgBox "Workbook synchronization failed. " & _  
            "Please contact your administrator " & vbCrLf & _  
            "or try again later."  
  
    End If  
  
End Sub
```



[Show All](#)

# Microsoft Excel Constants

This topic lists all constants in the Microsoft Excel object model.

## [xlApplicationInternational](#)

Constant	Value
xl24HourClock	33
xl4DigitYears	43
xlAlternateArraySeparator	16
xlColumnSeparator	14
xlCountryCode	1
xlCountrySetting	2
xlCurrencyBefore	37
xlCurrencyCode	25
xlCurrencyDigits	27
xlCurrencyLeadingZeros	40
xlCurrencyMinusSign	38
xlCurrencyNegative	28
xlCurrencySpaceBefore	36
xlCurrencyTrailingZeros	39
xlDateOrder	32
xlDateSeparator	17
xlDayCode	21
xlDayLeadingZero	42
xlDecimalSeparator	3
xlGeneralFormatName	26
xlHourCode	22
xlLeftBrace	12
xlLeftBracket	10
xlListSeparator	5
xlLowerCaseColumnLetter	9

xlLowerCaseRowLetter	8
xlMDY	44
xlMetric	35
xlMinuteCode	23
xlMonthCode	20
xlMonthLeadingZero	41
xlMonthNameChars	30
xlNoncurrencyDigits	29
xlNonEnglishFunctions	34
xlRightBrace	13
xlRightBracket	11
xlRowSeparator	15
xlSecondCode	24
xlThousandsSeparator	4
xlTimeLeadingZero	45
xlTimeSeparator	18
xlUpperCaseColumnLetter	7
xlUpperCaseRowLetter	6
xlWeekdayNameChars	31
xlYearCode	19

## [xlApplyNamesOrder](#)

<b>Constant</b>	<b>Value</b>
xlColumnThenRow	2
xlRowThenColumn	1

## [xlArabicModes](#)

<b>Constant</b>	<b>Value</b>
xlArabicBothStrict	3
xlArabicNone	0
xlArabicStrictAlefHamza	1
xlArabicStrictFinalYaa	2

## [xlArrangeStyle](#)

<b>Constant</b>	<b>Value</b>
xlArrangeStyleCascade	7
xlArrangeStyleHorizontal	-4128
xlArrangeStyleTiled	1
xlArrangeStyleVertical	-4166

## [xlArrowHeadLength](#)

<b>Constant</b>	<b>Value</b>
xlArrowHeadLengthLong	3
xlArrowHeadLengthMedium	-4138
xlArrowHeadLengthShort	1

## [xlArrowHeadStyle](#)

<b>Constant</b>	<b>Value</b>
xlArrowHeadStyleClosed	3
xlArrowHeadStyleDoubleClosed	5
xlArrowHeadStyleDoubleOpen	4
xlArrowHeadStyleNone	-4142
xlArrowHeadStyleOpen	2

## [xlArrowHeadWidth](#)

<b>Constant</b>	<b>Value</b>
xlArrowHeadWidthMedium	-4138
xlArrowHeadWidthNarrow	1
xlArrowHeadWidthWide	3

## [xlAutoFillType](#)

<b>Constant</b>	<b>Value</b>
xlFillCopy	1
xlFillDays	5
xlFillDefault	0
xlFillFormats	3
xlFillMonths	7
xlFillSeries	2
xlFillValues	4
xlFillWeekdays	6
xlFillYears	8
xlGrowthTrend	10
xlLinearTrend	9

### [xlAutoFilterOperator](#)

<b>Constant</b>	<b>Value</b>
xlAnd	1
xlBottom10Items	4
xlBottom10Percent	6
xlOr	2
xlTop10Items	3
xlTop10Percent	5

### [xlAxisCrosses](#)

<b>Constant</b>	<b>Value</b>
xlAxisCrossesAutomatic	-4105
xlAxisCrossesCustom	-4114
xlAxisCrossesMaximum	2
xlAxisCrossesMinimum	4

### [xlAxisGroup](#)

**Constant Value**

xlPrimary 1  
xlSecondary 2

### [xlAxisType](#)

<b>Constant</b>	<b>Value</b>
xlCategory	1
xlSeriesAxis	3
xlValue	2

### [xlBackground](#)

<b>Constant</b>	<b>Value</b>
xlBackgroundAutomatic	-4105
xlBackgroundOpaque	3
xlBackgroundTransparent	2

### [xlBarShape](#)

<b>Constant</b>	<b>Value</b>
xlBox	0
xlConeToMax	5
xlConeToPoint	4
xlCylinder	3
xlPyramidToMax	2
xlPyramidToPoint	1

### [xlBordersIndex](#)

<b>Constant</b>	<b>Value</b>
xlDiagonalDown	5
xlDiagonalUp	6
xlEdgeBottom	9
xlEdgeLeft	7

xlEdgeRight 10  
 xlEdgeTop 8  
 xlInsideHorizontal 12  
 xlInsideVertical 11

 [\*\*xlBorderWeight\*\*](#)

**Constant Value**

xlHairline 1  
 xlMedium -4138  
 xlThick 4  
 xlThin 2

 [\*\*xlBuiltInDialog\*\*](#)

<b>Constant</b>	<b>Value</b>
_xlDialogChartData	541
_xlDialogPhonetic	538
xlDialogActivate	103
xlDialogActiveCellFont	476
xlDialogAddChartAutoformat	390
xlDialogAddinManager	321
xlDialogAlignment	43
xlDialogApplyNames	133
xlDialogApplyStyle	212
xlDialogAppMove	170
xlDialogAppSize	171
xlDialogArrangeAll	12
xlDialogAssignToObject	213
xlDialogAssignToTool	293
xlDialogAttachText	80
xlDialogAttachToolbars	323
xlDialogAutoCorrect	485
xlDialogAxes	78

xlDialogBorder	45
xlDialogCalculation	32
xlDialogCellProtection	46
xlDialogChangeLink	166
xlDialogChartAddData	392
xlDialogChartLocation	527
xlDialogChartOptionsDataLabelMultiple	724
xlDialogChartOptionsDataLabels	505
xlDialogChartOptionsDataTable	506
xlDialogChartSourceData	540
xlDialogChartTrend	350
xlDialogChartType	526
xlDialogChartWizard	288
xlDialogCheckboxProperties	435
xlDialogClear	52
xlDialogColorPalette	161
xlDialogColumnWidth	47
xlDialogCombination	73
xlDialogConditionalFormatting	583
xlDialogConsolidate	191
xlDialogCopyChart	147
xlDialogCopyPicture	108
xlDialogCreateList	796
xlDialogCreateNames	62
xlDialogCreatePublisher	217
xlDialogCustomizeToolbar	276
xlDialogCustomViews	493
xlDialogDataDelete	36
xlDialogDataLabel	379
xlDialogDataLabelMultiple	723
xlDialogDataSeries	40
xlDialogDataValidation	525
xlDialogDefineName	61

xlDialogDefineStyle	229
xlDialogDeleteFormat	111
xlDialogDeleteName	110
xlDialogDemote	203
xlDialogDisplay	27
xlDialogEditboxProperties	438
xlDialogEditColor	223
xlDialogEditDelete	54
xlDialogEditionOptions	251
xlDialogEditSeries	228
xlDialogErrorbarX	463
xlDialogErrorbarY	464
xlDialogErrorChecking	732
xlDialogEvaluateFormula	709
xlDialogExternalDataProperties	530
xlDialogExtract	35
xlDialogFileDelete	6
xlDialogFileSharing	481
xlDialogFillGroup	200
xlDialogFillWorkgroup	301
xlDialogFilter	447
xlDialogFilterAdvanced	370
xlDialogFindFile	475
xlDialogFont	26
xlDialogFontProperties	381
xlDialogFormatAuto	269
xlDialogFormatChart	465
xlDialogFormatCharttype	423
xlDialogFormatFont	150
xlDialogFormatLegend	88
xlDialogFormatMain	225
xlDialogFormatMove	128
xlDialogFormatNumber	42

xlDialogFormatOverlay	226
xlDialogFormatSize	129
xlDialogFormatText	89
xlDialogFormulaFind	64
xlDialogFormulaGoto	63
xlDialogFormulaReplace	130
xlDialogFunctionWizard	450
xlDialogGallery3dArea	193
xlDialogGallery3dBar	272
xlDialogGallery3dColumn	194
xlDialogGallery3dLine	195
xlDialogGallery3dPie	196
xlDialogGallery3dSurface	273
xlDialogGalleryArea	67
xlDialogGalleryBar	68
xlDialogGalleryColumn	69
xlDialogGalleryCustom	388
xlDialogGalleryDoughnut	344
xlDialogGalleryLine	70
xlDialogGalleryPie	71
xlDialogGalleryRadar	249
xlDialogGalleryScatter	72
xlDialogGoalSeek	198
xlDialogGridlines	76
xlDialogImportTextFile	666
xlDialogInsert	55
xlDialogInsertHyperlink	596
xlDialogInsertNameLabel	496
xlDialogInsertObject	259
xlDialogInsertPicture	342
xlDialogInsertTitle	380
xlDialogLabelProperties	436
xlDialogListboxProperties	437

xlDialogMacroOptions	382
xlDialogMailEditMailer	470
xlDialogMailLogon	339
xlDialogMailNextLetter	378
xlDialogMainChart	85
xlDialogMainChartType	185
xlDialogMenuEditor	322
xlDialogMove	262
xlDialogMyPermission	834
xlDialogNew	119
xlDialogNewWebQuery	667
xlDialogNote	154
xlDialogObjectProperties	207
xlDialogObjectProtection	214
xlDialogOpen	1
xlDialogOpenLinks	2
xlDialogOpenMail	188
xlDialogOpenText	441
xlDialogOptionsCalculation	318
xlDialogOptionsChart	325
xlDialogOptionsEdit	319
xlDialogOptionsGeneral	356
xlDialogOptionsListsAdd	458
xlDialogOptionsME	647
xlDialogOptionsTransition	355
xlDialogOptionsView	320
xlDialogOutline	142
xlDialogOverlay	86
xlDialogOverlayChartType	186
xlDialogPageSetup	7
xlDialogParse	91
xlDialogPasteNames	58
xlDialogPasteSpecial	53

xlDialogPatterns	84
xlDialogPermission	832
xlDialogPhonetic	656
xlDialogPivotCalculatedField	570
xlDialogPivotCalculatedItem	572
xlDialogPivotClientServerSet	689
xlDialogPivotFieldGroup	433
xlDialogPivotFieldProperties	313
xlDialogPivotFieldUngroup	434
xlDialogPivotShowPages	421
xlDialogPivotSolveOrder	568
xlDialogPivotTableOptions	567
xlDialogPivotTableWizard	312
xlDialogPlacement	300
xlDialogPrint	8
xlDialogPrinterSetup	9
xlDialogPrintPreview	222
xlDialogPromote	202
xlDialogProperties	474
xlDialogPropertyFields	754
xlDialogProtectDocument	28
xlDialogProtectSharing	620
xlDialogPublishAsWebPage	653
xlDialogPushbuttonProperties	445
xlDialogReplaceFont	134
xlDialogRoutingSlip	336
xlDialogRowHeight	127
xlDialogRun	17
xlDialogSaveAs	5
xlDialogSaveCopyAs	456
xlDialogSaveNewObject	208
xlDialogSaveWorkbook	145
xlDialogSaveWorkspace	285

xlDialogScale	87
xlDialogScenarioAdd	307
xlDialogScenarioCells	305
xlDialogScenarioEdit	308
xlDialogScenarioMerge	473
xlDialogScenarioSummary	311
xlDialogScrollbarProperties	420
xlDialogSearch	731
xlDialogSelectSpecial	132
xlDialogSendMail	189
xlDialogSeriesAxes	460
xlDialogSeriesOptions	557
xlDialogSeriesOrder	466
xlDialogSeriesShape	504
xlDialogSeriesX	461
xlDialogSeriesY	462
xlDialogSetBackgroundPicture	509
xlDialogSetPrintTitles	23
xlDialogSetUpdateStatus	159
xlDialogShowDetail	204
xlDialogShowToolbar	220
xlDialogSize	261
xlDialogSort	39
xlDialogSortSpecial	192
xlDialogSplit	137
xlDialogStandardFont	190
xlDialogStandardWidth	472
xlDialogStyle	44
xlDialogSubscribeTo	218
xlDialogSubtotalCreate	398
xlDialogSummaryInfo	474
xlDialogTable	41
xlDialogTabOrder	394

xlDialogTextToColumns	422
xlDialogUnhide	94
xlDialogUpdateLink	201
xlDialogVbaInsertFile	328
xlDialogVbaMakeAddin	478
xlDialogVbaProcedureDefinition	330
xlDialogView3d	197
xlDialogWebOptionsBrowsers	773
xlDialogWebOptionsEncoding	686
xlDialogWebOptionsFiles	684
xlDialogWebOptionsFonts	687
xlDialogWebOptionsGeneral	683
xlDialogWebOptionsPictures	685
xlDialogWindowMove	14
xlDialogWindowSize	13
xlDialogWorkbookAdd	281
xlDialogWorkbookCopy	283
xlDialogWorkbookInsert	354
xlDialogWorkbookMove	282
xlDialogWorkbookName	386
xlDialogWorkbookNew	302
xlDialogWorkbookOptions	284
xlDialogWorkbookProtect	417
xlDialogWorkbookTabSplit	415
xlDialogWorkbookUnhide	384
xlDialogWorkgroup	199
xlDialogWorkspace	95
xlDialogZoom	256

## [xlCalculatedMemberType](#)

Constant	Value
xlCalculatedMember 0	

xlCalculatedSet 1

## [xlCalculation](#)

Constant	Value
xlCalculationAutomatic	-4105
xlCalculationManual	-4135
xlCalculationSemiautomatic	2

## [xlCalculationInterruptKey](#)

### Constant Value

xlAnyKey	2
xlEscKey	1
xlNoKey	0

## [xlCalculationState](#)

Constant	Value
xlCalculating	1
xlDone	0
xlPending	2

## [xlCategoryType](#)

Constant	Value
xlAutomaticScale	-4105
xlCategoryScale	2
xlTimeScale	3

## [xlCellInsertionMode](#)

Constant	Value
xlInsertDeleteCells	1

xlInsertEntireRows 2  
xlOverwriteCells 0

## [xlCellType](#)

<b>Constant</b>	<b>Value</b>
xlCellTypeAllFormatConditions	-4172
xlCellTypeAllValidation	-4174
xlCellTypeBlanks	4
xlCellTypeComments	-4144
xlCellTypeConstants	2
xlCellTypeFormulas	-4123
xlCellTypeLastCell	11
xlCellTypeSameFormatConditions	-4173
xlCellTypeSameValidation	-4175
xlCellTypeVisible	12

## [xlChartGallery](#)

<b>Constant</b>	<b>Value</b>
xlAnyGallery	23
xlBuiltIn	21
xlUserDefined	22

## [xlChartItem](#)

<b>Constant</b>	<b>Value</b>
xlAxis	21
xlAxisTitle	17
xlChartArea	2
xlChartTitle	4
xlCorners	6
xlDataLabel	0
xlDataTable	7

xlDisplayUnitLabel	30
xlDownBars	20
xlDropLines	26
xlErrorBars	9
xlFloor	23
xlHiLoLines	25
xlLeaderLines	29
xlLegend	24
xlLegendEntry	12
xlLegendKey	13
xlMajorGridlines	15
xlMinorGridlines	16
xlNothing	28
xlPivotChartDropZone	32
xlPivotChartFieldButton	31
xlPlotArea	19
xlRadarAxisLabels	27
xlSeries	3
xlSeriesLines	22
xlShape	14
xlTrendline	8
xlUpBars	18
xlWalls	5
xlXErrorBars	10
xlYErrorBars	11

 [\*\*xlChartLocation\*\*](#)

<b>Constant</b>	<b>Value</b>
xlLocationAsNewSheet	1
xlLocationAsObject	2
xlLocationAutomatic	3

## [xlChartPicturePlacement](#)

<b>Constant</b>	<b>Value</b>
xlAllFaces	7
xlEnd	2
xlEndSides	3
xlFront	4
xlFrontEnd	6
xlFrontSides	5
xlSides	1

## [xlChartPictureType](#)

<b>Constant</b>	<b>Value</b>
xlStack	2
xlStackScale	3
xlStretch	1

## [xlChartSplitType](#)

<b>Constant</b>	<b>Value</b>
xlSplitByCustomSplit	4
xlSplitByPercentValue	3
xlSplitByPosition	1
xlSplitByValue	2

## [xlChartType](#)

<b>Constant</b>	<b>Value</b>
xl3DArea	-4098
xl3DAreaStacked	78
xl3DAreaStacked100	79
xl3DBarClustered	60
xl3DBarStacked	61

xl3DBarStacked100	62
xl3DColumn	-4100
xl3DColumnClustered	54
xl3DColumnStacked	55
xl3DColumnStacked100	56
xl3DLine	-4101
xl3DPie	-4102
xl3DPieExploded	70
xlArea	1
xlAreaStacked	76
xlAreaStacked100	77
xlBarClustered	57
xlBarOfPie	71
xlBarStacked	58
xlBarStacked100	59
xlBubble	15
xlBubble3DEffect	87
xlColumnClustered	51
xlColumnStacked	52
xlColumnStacked100	53
xlConeBarClustered	102
xlConeBarStacked	103
xlConeBarStacked100	104
xlConeCol	105
xlConeColClustered	99
xlConeColStacked	100
xlConeColStacked100	101
xlCylinderBarClustered	95
xlCylinderBarStacked	96
xlCylinderBarStacked100	97
xlCylinderCol	98
xlCylinderColClustered	92
xlCylinderColStacked	93

xlCylinderColStacked100	94
xlDoughnut	-4120
xlDoughnutExploded	80
xlLine	4
xlLineMarkers	65
xlLineMarkersStacked	66
xlLineMarkersStacked100	67
xlLineStacked	63
xlLineStacked100	64
xlPie	5
xlPieExploded	69
xlPieOfPie	68
xlPyramidBarClustered	109
xlPyramidBarStacked	110
xlPyramidBarStacked100	111
xlPyramidCol	112
xlPyramidColClustered	106
xlPyramidColStacked	107
xlPyramidColStacked100	108
xlRadar	-4151
xlRadarFilled	82
xlRadarMarkers	81
xlStockHLC	88
xlStockOHLC	89
xlStockVHLC	90
xlStockVOHLC	91
xlSurface	83
xlSurfaceTopView	85
xlSurfaceTopViewWireframe	86
xlSurfaceWireframe	84
xlXYScatter	-4169
xlXYScatterLines	74
xlXYScatterLinesNoMarkers	75

xlXYScatterSmooth	72
xlXYScatterSmoothNoMarkers	73

## [xlClipboardFormat](#)

<b>Constant</b>	<b>Value</b>
xlClipboardFormatBIFF	8
xlClipboardFormatBIFF2	18
xlClipboardFormatBIFF3	20
xlClipboardFormatBIFF4	30
xlClipboardFormatBinary	15
xlClipboardFormatBitmap	9
xlClipboardFormatCGM	13
xlClipboardFormatCSV	5
xlClipboardFormatDIF	4
xlClipboardFormatDspText	12
xlClipboardFormatEmbeddedObject	21
xlClipboardFormatEmbedSource	22
xlClipboardFormatLink	11
xlClipboardFormatLinkSource	23
xlClipboardFormatLinkSourceDesc	32
xlClipboardFormatMovie	24
xlClipboardFormatNative	14
xlClipboardFormatObjectDesc	31
xlClipboardFormatObjectLink	19
xlClipboardFormatOwnerLink	17
xlClipboardFormatPICT	2
xlClipboardFormatPrintPICT	3
xlClipboardFormatRTF	7
xlClipboardFormatScreenPICT	29
xlClipboardFormatStandardFont	28
xlClipboardFormatStandardScale	27
xlClipboardFormatSYLK	6

xlClipboardFormatTable	16
xlClipboardFormatText	0
xlClipboardFormatToolFace	25
xlClipboardFormatToolFacePICT	26
xlClipboardFormatVALU	1
xlClipboardFormatWK1	10

## [xlCmdType](#)

Constant	Value
xlCmdCube	1
xlCmdDefault	4
xlCmdList	5
xlCmdSql	2
xlCmdTable	3

## [xlColorIndex](#)

Constant	Value
xlColorIndexAutomatic	-4105
xlColorIndexNone	-4142

## [xlColumnDataType](#)

Constant	Value
xlDMYFormat	4
xlDYMFormat	7
xlEMDFormat	10
xlGeneralFormat	1
xlMDYFormat	3
xlMYDFormat	6
xlSkipColumn	9
xlTextFormat	2
xlYDMFormat	8

xlYMDFormat 5

### [xlCommandUnderlines](#)

<b>Constant</b>	<b>Value</b>
xlCommandUnderlinesAutomatic	-4105
xlCommandUnderlinesOff	-4146
xlCommandUnderlinesOn	1

### [xlCommentDisplayMode](#)

<b>Constant</b>	<b>Value</b>
xlCommentAndIndicator	1
xlCommentIndicatorOnly	-1
xlNoIndicator	0

### [xlConsolidationFunction](#)

<b>Constant</b>	<b>Value</b>
xlAverage	-4106
xlCount	-4112
xlCountNums	-4113
xlMax	-4136
xlMin	-4139
xlProduct	-4149
xlStDev	-4155
xlStDevP	-4156
xlSum	-4157
xlUnknown	1000
xlVar	-4164
xlVarP	-4165

### [xlCopyPictureFormat](#)

### **Constant Value**

xlBitmap 2

xlPicture -4147

### [xlCorruptLoad](#)

#### **Constant Value**

xlExtractData 2

xlNormalLoad 0

xlRepairFile 1

### [xlCreator](#)

#### **Constant Value**

xlCreatorCode 1480803660

### [xlCubeFieldType](#)

#### **Constant Value**

xlHierarchy 1

xlMeasure 2

xlSet 3

### [xlCutCopyMode](#)

#### **Constant Value**

xlCopy 1

xlCut 2

### [xlCVError](#)

#### **Constant Value**

xlErrDiv0 2007

xlErrNA 2042

xlErrName 2029  
xlErrNull 2000  
xlErrNum 2036  
xlErrRef 2023  
xlErrValue 2015

### [xlDataLabelPosition](#)

<b>Constant</b>	<b>Value</b>
xlLabelPositionAbove	0
xlLabelPositionBelow	1
xlLabelPositionBestFit	5
xlLabelPositionCenter	-4108
xlLabelPositionCustom	7
xlLabelPositionInsideBase	4
xlLabelPositionInsideEnd	3
xlLabelPositionLeft	-4131
xlLabelPositionMixed	6
xlLabelPositionOutsideEnd	2
xlLabelPositionRight	-4152

### [xlDataLabelSeparator](#)

<b>Constant</b>	<b>Value</b>
xlDataLabelSeparatorDefault	1

### [xlDataLabelsType](#)

<b>Constant</b>	<b>Value</b>
xlDataLabelsShowBubbleSizes	6
xlDataLabelsShowLabel	4
xlDataLabelsShowLabelAndPercent	5
xlDataLabelsShowNone	-4142
xlDataLabelsShowPercent	3

xlDataLabelsShowValue 2

[xlDataSeriesDate](#)

**Constant Value**

xlDay 1  
xlMonth 3  
xlWeekday 2  
xlYear 4

[xlDataSeriesType](#)

**Constant Value**

xlAutoFill 4  
xlChronological 3  
xlDataSeriesLinear -4132  
xlGrowth 2

[xlDeleteShiftDirection](#)

**Constant Value**

xlShiftToLeft -4159  
xlShiftUp -4162

[xlDirection](#)

**Constant Value**

xlDown -4121  
xlToLeft -4159  
xlToRight -4161  
xlUp -4162

[xlDisplayBlanksAs](#)

<b>Constant</b>	<b>Value</b>
-----------------	--------------

xlInterpolated	3
xlNotPlotted	1
xlZero	2

<input type="checkbox"/>	<a href="#"><b>XIDisplayDrawingObjects</b></a>
--------------------------	--

<b>Constant</b>	<b>Value</b>
-----------------	--------------

xlDisplayShapes	-4104
xlHide	3
xlPlaceholders	2

<input type="checkbox"/>	<a href="#"><b>XIDisplayUnit</b></a>
--------------------------	--------------------------------------

<b>Constant</b>	<b>Value</b>
-----------------	--------------

xlHundredMillions	-8
xlHundreds	-2
xlHundredThousands	-5
xlMillionMillions	-10
xlMillions	-6
xlTenMillions	-7
xlTenThousands	-4
xlThousandMillions	-9
xlThousands	-3

<input type="checkbox"/>	<a href="#"><b>XIDVAlertStyle</b></a>
--------------------------	---------------------------------------

<b>Constant</b>	<b>Value</b>
-----------------	--------------

xlValidAlertInformation	3
xlValidAlertStop	1
xlValidAlertWarning	2

<input type="checkbox"/>	<a href="#"><b>XIDVType</b></a>
--------------------------	---------------------------------

<b>Constant</b>	<b>Value</b>
xlValidateCustom	7
xlValidateDate	4
xlValidateDecimal	2
xlValidateInputOnly	0
xlValidateList	3
xlValidateTextLength	6
xlValidateTime	5
xlValidateWholeNumber	1

### [XlEditionFormat](#)

#### **Constant Value**

xlBIFF	2
xlPICT	1
xlRTF	4
xlVALU	8

### [XlEditionOptionsOption](#)

<b>Constant</b>	<b>Value</b>
xlAutomaticUpdate	4
xlCancel	1
xlChangeAttributes	6
xlManualUpdate	5
xlOpenSource	3
xlSelect	3
xlSendPublisher	2
xlUpdateSubscriber	2

### [XlEditionType](#)

#### **Constant Value**

xlPublisher	1
-------------	---

xlSubscriber 2

 [xlEnableCancelKey](#)

<b>Constant</b>	<b>Value</b>
xlDisabled	0
xlErrorHandler	2
xlInterrupt	1

 [xlEnableSelection](#)

<b>Constant</b>	<b>Value</b>
xlNoRestrictions	0
xlNoSelection	-4142
xlUnlockedCells	1

 [xlEndStyleCap](#)

**Constant Value**

xlCap	1
xlNoCap	2

 [xlErrorBarDirection](#)

**Constant Value**

xlX	-4168
xlY	1

 [xlErrorBarInclude](#)

<b>Constant</b>	<b>Value</b>
xlErrorBarIncludeBoth	1
xlErrorBarIncludeMinusValues	3
xlErrorBarIncludeNone	-4142

xlErrorBarIncludePlusValues 2

## [xlErrorBarType](#)

<b>Constant</b>	<b>Value</b>
xlErrorBarTypeCustom	-4114
xlErrorBarTypeFixedValue	1
xlErrorBarTypePercent	2
xlErrorBarTypeStDev	-4155
xlErrorBarTypeStError	4

## [xlErrorChecks](#)

<b>Constant</b>	<b>Value</b>
xlEmptyCellReferences	7
xlEvaluateToError	1
xlInconsistentFormula	4
xlListDataValidation	8
xlNumberAsText	3
xlOmittedCells	5
xlTextDate	2
xlUnlockedFormulaCells	6

## [xlFileAccess](#)

<b>Constant</b>	<b>Value</b>
xlReadOnly	3
xlReadWrite	2

## [xlFileFormat](#)

<b>Constant</b>	<b>Value</b>
xlAddIn	18
xlCSV	6

xlCSVMac	22
xlCSVMSDOS	24
xlCSVWindows	23
xlCurrentPlatformText	-4158
xlDBF2	7
xlDBF3	8
xlDBF4	11
xlDIF	9
xlExcel2	16
xlExcel2FarEast	27
xlExcel3	29
xlExcel4	33
xlExcel4Workbook	35
xlExcel5	39
xlExcel7	39
xlExcel9795	43
xlHtml	44
xlIntlAddIn	26
xlIntlMacro	25
xlSYLK	2
xlTemplate	17
xlTextMac	19
xlTextMSDOS	21
xlTextPrinter	36
xlTextWindows	20
xlUnicodeText	42
xlWebArchive	45
xlWJ2WD1	14
xlWJ3	40
xlWJ3FJ3	41
xlWK1	5
xlWK1ALL	31
xlWK1FMT	30

xlWK3	15
xlWK3FM3	32
xlWK4	38
xlWKS	4
xlWorkbookNormal	-4143
xlWorks2FarEast	28
xlWQ1	34
xlXMLData	47
xlXMLSpreadsheet	46

### [xlFillWith](#)

Constant	Value
xlFillWithAll	-4104
xlFillWithContents	2
xlFillWithFormats	-4122

### [xlFilterAction](#)

Constant	Value
xlFilterCopy	2
xlFilterInPlace	1

### [xlFindLookIn](#)

Constant	Value
xlComments	-4144
xlFormulas	-4123
xlValues	-4163

### [xlFormatConditionOperator](#)

Constant	Value
xlBetween	1

xlEqual	3
xlGreater	5
xlGreaterEqual	7
xlLess	6
xlLessEqual	8
xlNotBetween	2
xlNotEqual	4

 [xlFormatConditionType](#)

**Constant Value**

xlCellValue	1
xlExpression	2

 [xlFormControl](#)

**Constant Value**

xlButtonControl	0
xlCheckBox	1
xlDropDown	2
xlEditBox	3
xlGroupBox	4
xlLabel	5
xlListBox	6
xlOptionButton	7
xlScrollBar	8
xlSpinner	9

 [xlFormulaLabel](#)

**Constant Value**

xlColumnLabels	2
xlMixedLabels	3
xlNoLabels	-4142

xlRowLabels 1

## [xlHAlign](#)

<b>Constant</b>	<b>Value</b>
xlHAlignCenter	-4108
xlHAlignCenterAcrossSelection	7
xlHAlignDistributed	-4117
xlHAlignFill	5
xlHAlignGeneral	1
xlHAlignJustify	-4130
xlHAlignLeft	-4131
xlHAlignRight	-4152

## [xlHebrewModes](#)

<b>Constant</b>	<b>Value</b>
xlHebrewFullScript	0
xlHebrewMixedAuthorizedScript	3
xlHebrewMixedScript	2
xlHebrewPartialScript	1

## [xlHighlightChangesTime](#)

<b>Constant</b>	<b>Value</b>
xlAllChanges	2
xlNotYetReviewed	3
xlSinceMyLastSave	1

## [xlHtmlType](#)

<b>Constant</b>	<b>Value</b>
xlHtmlCalc	1
xlHtmlChart	3

xlHtmlList 2  
xlHtmlStatic 0

### [xlIMEMode](#)

<b>Constant</b>	<b>Value</b>
xlIMEModeAlpha	8
xlIMEModeAlphaFull	7
xlIMEModeDisable	3
xlIMEModeHangul	10
xlIMEModeHangulFull	9
xlIMEModeHiragana	4
xlIMEModeKatakana	5
xlIMEModeKatakanaHalf	6
xlIMEModeNoControl	0
xlIMEModeOff	2
xlIMEModeOn	1

### [xlImportDataAs](#)

<b>Constant</b>	<b>Value</b>
xlPivotTableReport	1
xlQueryTable	0

### [xlInsertFormatOrigin](#)

<b>Constant</b>	<b>Value</b>
xlFormatFromLeftOrAbove	0
xlFormatFromRightOrBelow	1

### [xlInsertShiftDirection](#)

<b>Constant</b>	<b>Value</b>
xlShiftDown	-4121

xlShiftToRight -4161

## [XLLayoutFormType](#)

### **Constant Value**

xlOutline 1

xlTabular 0

## [XLLegendPosition](#)

<b>Constant</b>	<b>Value</b>
xlLegendPositionBottom	-4107
xlLegendPositionCorner	2
xlLegendPositionLeft	-4131
xlLegendPositionRight	-4152
xlLegendPositionTop	-4160

## [XLLineStyle](#)

<b>Constant</b>	<b>Value</b>
xlContinuous	1
xlDash	-4115
xlDashDot	4
xlDashDotDot	5
xlDot	-4118
xlDouble	-4119
xlLineStyleNone	-4142
xlSlantDashDot	13

## [XLLink](#)

### **Constant Value**

xlExcelLinks 1

xlOLELinks 2

xlPublishers 5  
xlSubscribers 6

### [XlLinkInfo](#)

<b>Constant</b>	<b>Value</b>
xlEditionDate	2
xlLinkInfoStatus	3
xlUpdateState	1

### [XlLinkInfoType](#)

<b>Constant</b>	<b>Value</b>
xlLinkInfoOLELinks	2
xlLinkInfoPublishers	5
xlLinkInfoSubscribers	6

### [XlLinkStatus](#)

<b>Constant</b>	<b>Value</b>
xlLinkStatusCopiedValues	10
xlLinkStatusIndeterminate	5
xlLinkStatusInvalidName	7
xlLinkStatusMissingFile	1
xlLinkStatusMissingSheet	2
xlLinkStatusNotStarted	6
xlLinkStatusOK	0
xlLinkStatusOld	3
xlLinkStatusSourceNotCalculated	4
xlLinkStatusSourceNotOpen	8
xlLinkStatusSourceOpen	9

### [XlLinkType](#)

<b>Constant</b>	<b>Value</b>
xlLinkTypeExcelLinks	1
xlLinkTypeOLELinks	2

### [\*\*XIListConflict\*\*](#)

<b>Constant</b>	<b>Value</b>
xlListConflictDialog	0
xlListConflictDiscardAllConflicts	2
xlListConflictError	3
xlListConflictRetryAllConflicts	1

### [\*\*XIListDataType\*\*](#)

<b>Constant</b>	<b>Value</b>
xlListDataTypeCheckbox	9
xlListDataTypeChoice	6
xlListDataTypeChoiceMulti	7
xlListDataTypeCounter	11
xlListDataTypeCurrency	4
xlListDataTypeDateTime	5
xlListDataTypeHyperLink	10
xlListDataTypeListLookup	8
xlListDataTypeMultiLineRichText	12
xlListDataTypeMultiLineText	2
xlListDataTypeNone	0
xlListDataTypeNumber	3
xlListDataTypeText	1

### [\*\*XIListObjectSourceType\*\*](#)

<b>Constant</b>	<b>Value</b>
xlSrcExternal	0
xlSrcRange	1

xlSrcXml 2

## [xlLocationInTable](#)

<b>Constant</b>	<b>Value</b>
xlColumnHeader	-4110
xlColumnItem	5
xlDataHeader	3
xlDataItem	7
xlPageHeader	2
xlPageItem	6
xlRowHeader	-4153
xlRowItem	4
xlTableBody	8

## [xlLookAt](#)

### **Constant Value**

xlPart	2
xlWhole	1

## [xlMailSystem](#)

<b>Constant</b>	<b>Value</b>
xlMAPI	1
xlNoMailSystem	0
xlPowerTalk	2

## [xlMarkerStyle](#)

<b>Constant</b>	<b>Value</b>
xlMarkerStyleAutomatic	-4105
xlMarkerStyleCircle	8
xlMarkerStyleDash	-4115

xlMarkerStyleDiamond	2
xlMarkerStyleDot	-4118
xlMarkerStyleNone	-4142
xlMarkerStylePicture	-4147
xlMarkerStylePlus	9
xlMarkerStyleSquare	1
xlMarkerStyleStar	5
xlMarkerStyleTriangle	3
xlMarkerStyleX	-4168

### [xlMouseButton](#)

<b>Constant</b>	<b>Value</b>
xlNoButton	0
xlPrimaryButton	1
xlSecondaryButton	2

### [xlMousePointer](#)

<b>Constant</b>	<b>Value</b>
xlDefault	-4143
xlIBeam	3
xlNorthwestArrow	1
xlWait	2

### [xlMSApplication](#)

<b>Constant</b>	<b>Value</b>
xlMicrosoftAccess	4
xlMicrosoftFoxPro	5
xlMicrosoftMail	3
xlMicrosoftPowerPoint	2
xlMicrosoftProject	6
xlMicrosoftSchedulePlus	7

xlMicrosoftWord 1

### [xlObjectSize](#)

#### **Constant Value**

xlFitToPage 2

xlFullPage 3

xlScreenSize 1

### [xlOLEType](#)

#### **Constant Value**

xlOLEControl 2

xlOLEEmbed 1

xlOLELink 0

### [xlOLEVerb](#)

#### **Constant Value**

xlVerbOpen 2

xlVerbPrimary 1

### [xlOrder](#)

#### **Constant Value**

xlDownThenOver 1

xlOverThenDown 2

### [xlOrientation](#)

#### **Constant Value**

xlDownward -4170

xlHorizontal -4128

xlUpward -4171

xlVertical -4166

## [xlPageBreak](#)

Constant	Value
xlPageBreakAutomatic	-4105
xlPageBreakManual	-4135
xlPageBreakNone	-4142

## [xlPageBreakExtent](#)

Constant	Value
xlPageBreakFull	1
xlPageBreakPartial	2

## [xlPageOrientation](#)

Constant	Value
xlLandscape	2
xlPortrait	1

## [xlPaperSize](#)

Constant	Value
xlPaper10x14	16
xlPaper11x17	17
xlPaperA3	8
xlPaperA4	9
xlPaperA4Small	10
xlPaperA5	11
xlPaperB4	12
xlPaperB5	13
xlPaperCsheet	24
xlPaperDsheet	25

xlPaperEnvelope10	20
xlPaperEnvelope11	21
xlPaperEnvelope12	22
xlPaperEnvelope14	23
xlPaperEnvelope9	19
xlPaperEnvelopeB4	33
xlPaperEnvelopeB5	34
xlPaperEnvelopeB6	35
xlPaperEnvelopeC3	29
xlPaperEnvelopeC4	30
xlPaperEnvelopeC5	28
xlPaperEnvelopeC6	31
xlPaperEnvelopeC65	32
xlPaperEnvelopeDL	27
xlPaperEnvelopeItaly	36
xlPaperEnvelopeMonarch	37
xlPaperEnvelopePersonal	38
xlPaperEsheet	26
xlPaperExecutive	7
xlPaperFanfoldLegalGerman	41
xlPaperFanfoldStdGerman	40
xlPaperFanfoldUS	39
xlPaperFolio	14
xlPaperLedger	4
xlPaperLegal	5
xlPaperLetter	1
xlPaperLetterSmall	2
xlPaperNote	18
xlPaperQuarto	15
xlPaperStatement	6
xlPaperTabloid	3
xlPaperUser	256

## [xlParameterDataType](#)

<b>Constant</b>	<b>Value</b>
xlParamTypeBigInt	-5
xlParamTypeBinary	-2
xlParamTypeBit	-7
xlParamTypeChar	1
xlParamTypeDate	9
xlParamTypeDecimal	3
xlParamTypeDouble	8
xlParamTypeFloat	6
xlParamTypeInteger	4
xlParamTypeLongVarBinary	-4
xlParamTypeLongVarChar	-1
xlParamTypeNumeric	2
xlParamTypeReal	7
xlParamTypeSmallInt	5
xlParamTypeTime	10
xlParamTypeTimestamp	11
xlParamTypeTinyInt	-6
xlParamTypeUnknown	0
xlParamTypeVarBinary	-3
xlParamTypeVarChar	12
xlParamTypeWChar	-8

## [xlParameterType](#)

### **Constant Value**

xlConstant	1
xlPrompt	0
xlRange	2

## [xlPasteSpecialOperation](#)

<b>Constant</b>	<b>Value</b>
xlPasteSpecialOperationAdd	2
xlPasteSpecialOperationDivide	5
xlPasteSpecialOperationMultiply	4
xlPasteSpecialOperationNone	-4142
xlPasteSpecialOperationSubtract	3

## [XlPasteType](#)

<b>Constant</b>	<b>Value</b>
xlPasteAll	-4104
xlPasteAllExceptBorders	7
xlPasteColumnWidths	8
xlPasteComments	-4144
xlPasteFormats	-4122
xlPasteFormulas	-4123
xlPasteFormulasAndNumberFormats	11
xlPasteValidation	6
xlPasteValues	-4163
xlPasteValuesAndNumberFormats	12

## [XlPattern](#)

<b>Constant</b>	<b>Value</b>
xlPatternAutomatic	-4105
xlPatternChecker	9
xlPatternCrissCross	16
xlPatternDown	-4121
xlPatternGray16	17
xlPatternGray25	-4124
xlPatternGray50	-4125
xlPatternGray75	-4126
xlPatternGray8	18
xlPatternGrid	15

xlPatternHorizontal	-4128
xlPatternLightDown	13
xlPatternLightHorizontal	11
xlPatternLightUp	14
xlPatternLightVertical	12
xlPatternNone	-4142
xlPatternSemiGray75	10
xlPatternSolid	1
xlPatternUp	-4162
xlPatternVertical	-4166

### [xlPhoneticAlignment](#)

Constant	Value
xlPhoneticAlignCenter	2
xlPhoneticAlignDistributed	3
xlPhoneticAlignLeft	1
xlPhoneticAlignNoControl	0

### [xlPhoneticCharacterType](#)

Constant	Value
xlHiragana	2
xlKatakana	1
xlKatakanaHalf	0
xlNoConversion	3

### [xlPictureAppearance](#)

#### Constant Value

xlPrinter	2
xlScreen	1

### [xlPictureConvertorType](#)

## Constant Value

xlBMP	1
xlCGM	7
xlDRW	4
xlDXF	5
xlEPS	8
xlHGL	6
xlPCT	13
xlPCX	10
xlPIC	11
xlPLT	12
xlTIF	9
xlWMF	2
xlWPG	3

## [XlPivotCellType](#)

Constant	Value
xlPivotCellBlankCell	9
xlPivotCellCustomSubtotal	7
xlPivotCellDataField	4
xlPivotCellDataPivotField	8
xlPivotCellGrandTotal	3
xlPivotCellPageFieldItem	6
xlPivotCellPivotField	5
xlPivotCellPivotItem	1
xlPivotCellSubtotal	2
xlPivotCellValue	0

## [XlPivotFieldCalculation](#)

Constant	Value
xlDifferenceFrom	2

xlIndex	9
xlNoAdditionalCalculation	-4143
xlPercentDifferenceFrom	4
xlPercentOf	3
xlPercentOfColumn	7
xlPercentOfRow	6
xlPercentOfTotal	8
xlRunningTotal	5

### [XlPivotFieldType](#)

#### **Constant Value**

xlDate	2
xlNumber	-4145
xlText	-4158

### [XlPivotFieldOrientation](#)

#### **Constant Value**

xlColumnField	2
xlDataField	4
xlHidden	0
xlPageField	3
xlRowField	1

### [XlPivotFormatType](#)

#### **Constant Value**

xlPTClassic	20
xlPTNone	21
xlReport1	0
xlReport10	9
xlReport2	1
xlReport3	2

xlReport4	3
xlReport5	4
xlReport6	5
xlReport7	6
xlReport8	7
xlReport9	8
xlTable1	10
xlTable10	19
xlTable2	11
xlTable3	12
xlTable4	13
xlTable5	14
xlTable6	15
xlTable7	16
xlTable8	17
xlTable9	18

### [xlPivotTableMissingItems](#)

<b>Constant</b>	<b>Value</b>
xlMissingItemsDefault	-1
xlMissingItemsMax	32500
xlMissingItemsNone	0

### [xlPivotTableSourceType](#)

<b>Constant</b>	<b>Value</b>
xlConsolidation	3
xlDatabase	1
xlExternal	2
xlPivotTable	-4148
xlScenario	4

### [xlPivotTableVersionList](#)

<b>Constant</b>	<b>Value</b>
xlPivotTableVersion10	1
xlPivotTableVersion2000	0
xlPivotTableVersionCurrent	-1

### [xlPlacement](#)

<b>Constant</b>	<b>Value</b>
xlFreeFloating	3
xlMove	2
xlMoveAndSize	1

### [xlPlatform](#)

<b>Constant</b>	<b>Value</b>
xlMacintosh	1
xlMSDOS	3
xlWindows	2

### [xlPrintErrors](#)

<b>Constant</b>	<b>Value</b>
xlPrintErrorsBlank	1
xlPrintErrorsDash	2
xlPrintErrorsDisplayed	0
xlPrintErrorsNA	3

### [xlPrintLocation](#)

<b>Constant</b>	<b>Value</b>
xlPrintInPlace	16
xlPrintNoComments	-4142
xlPrintSheetEnd	1

## [xlPriority](#)

<b>Constant</b>	<b>Value</b>
xlPriorityHigh	-4127
xlPriorityLow	-4134
xlPriorityNormal	-4143

## [xlIPTSelectionMode](#)

<b>Constant</b>	<b>Value</b>
xlBlanks	4
xlButton	15
xlDataAndLabel	0
xlDataOnly	2
xlFirstRow	256
xlLabelOnly	1
xlOrigin	3

## [xlQueryType](#)

<b>Constant</b>	<b>Value</b>
xlADORecordset	7
xlDAORecordset	2
xlODBCQuery	1
xlOLEDBQuery	5
xlTextImport	6
xlWebQuery	4

## [xlRangeAutoFormat](#)

<b>Constant</b>	<b>Value</b>
xlRangeAutoFormat3DEffects1	13
xlRangeAutoFormat3DEffects2	14
xlRangeAutoFormatAccounting1	4

xlRangeAutoFormatAccounting2	5
xlRangeAutoFormatAccounting3	6
xlRangeAutoFormatAccounting4	17
xlRangeAutoFormatClassic1	1
xlRangeAutoFormatClassic2	2
xlRangeAutoFormatClassic3	3
xlRangeAutoFormatClassicPivotTable	31
xlRangeAutoFormatColor1	7
xlRangeAutoFormatColor2	8
xlRangeAutoFormatColor3	9
xlRangeAutoFormatList1	10
xlRangeAutoFormatList2	11
xlRangeAutoFormatList3	12
xlRangeAutoFormatLocalFormat1	15
xlRangeAutoFormatLocalFormat2	16
xlRangeAutoFormatLocalFormat3	19
xlRangeAutoFormatLocalFormat4	20
xlRangeAutoFormatNone	-4142
xlRangeAutoFormatPTNone	42
xlRangeAutoFormatReport1	21
xlRangeAutoFormatReport10	30
xlRangeAutoFormatReport2	22
xlRangeAutoFormatReport3	23
xlRangeAutoFormatReport4	24
xlRangeAutoFormatReport5	25
xlRangeAutoFormatReport6	26
xlRangeAutoFormatReport7	27
xlRangeAutoFormatReport8	28
xlRangeAutoFormatReport9	29
xlRangeAutoFormatSimple	-4154
xlRangeAutoFormatTable1	32
xlRangeAutoFormatTable10	41
xlRangeAutoFormatTable2	33

xlRangeAutoFormatTable3	34
xlRangeAutoFormatTable4	35
xlRangeAutoFormatTable5	36
xlRangeAutoFormatTable6	37
xlRangeAutoFormatTable7	38
xlRangeAutoFormatTable8	39
xlRangeAutoFormatTable9	40

## [xlRangeValueDataType](#)

Constant	Value
xlRangeValueDefault	10
xlRangeValueMSPersistXML	12
xlRangeValueXMLSpreadsheet	11

## [xlReferenceStyle](#)

### Constant Value

xlA1	1
xlR1C1	-4150

## [xlReferenceType](#)

Constant	Value
xlAbsolute	1
xlAbsRowRelColumn	2
xlRelative	4
xlRelRowAbsColumn	3

## [xlRobustConnect](#)

Constant	Value
xlAlways	1
xlAsRequired	0

xlNever 2

### [xlRoutingSlipDelivery](#)

Constant	Value
xlAllAtOnce	2
xlOneAfterAnother	1

### [xlRoutingSlipStatus](#)

Constant	Value
xlNotYetRouted	0
xlRoutingComplete	2
xlRoutingInProgress	1

### [xlRowCol](#)

Constant	Value
xlColumns	2
xlRows	1

### [xlRunAutoMacro](#)

Constant	Value
xlAutoActivate	3
xlAutoClose	2
xlAutoDeactivate	4
xlAutoOpen	1

### [xlSaveAction](#)

Constant	Value
xlDoNotSaveChanges	2
xlSaveChanges	1

## [xlSaveAsAccessMode](#)

<b>Constant</b>	<b>Value</b>
xlExclusive	3
xlNoChange	1
xlShared	2

## [xlSaveConflictResolution](#)

<b>Constant</b>	<b>Value</b>
xlLocalSessionChanges	2
xlOtherSessionChanges	3
xlUserResolution	1

## [xlScaleType](#)

<b>Constant</b>	<b>Value</b>
xlScaleLinear	-4132
xlScaleLogarithmic	-4133

## [xlSearchDirection](#)

<b>Constant</b>	<b>Value</b>
xlNext	1
xlPrevious	2

## [xlSearchOrder](#)

<b>Constant</b>	<b>Value</b>
xlByColumns	2
xlByRows	1

## [xlSearchWithin](#)

<b>Constant</b>	<b>Value</b>
xlWithinSheet	1
xlWithinWorkbook	2

### [xlSheetType](#)

<b>Constant</b>	<b>Value</b>
xlChart	-4109
xlDialogSheet	-4116
xlExcel4IntlMacroSheet	4
xlExcel4MacroSheet	3
xlWorksheet	-4167

### [xlSheetVisibility](#)

<b>Constant</b>	<b>Value</b>
xlSheetHidden	0
xlSheetVeryHidden	2
xlSheetVisible	-1

### [xlSizeRepresents](#)

<b>Constant</b>	<b>Value</b>
xlSizeIsArea	1
xlSizeIsWidth	2

### [xlSmartTagControlType](#)

<b>Constant</b>	<b>Value</b>
xlSmartTagControlActiveX	13
xlSmartTagControlButton	6
xlSmartTagControlCheckbox	9
xlSmartTagControlCombo	12
xlSmartTagControlHelp	3

xlSmartTagControlHelpURL	4
xlSmartTagControlImage	8
xlSmartTagControlLabel	7
xlSmartTagControlLink	2
xlSmartTagControlListbox	11
xlSmartTagControlRadioGroup	14
xlSmartTagControlSeparator	5
xlSmartTagControlSmartTag	1
xlSmartTagControlTextbox	10

### [xlSmartTagDisplayMode](#)

Constant	Value
xlButtonOnly	2
xlDisplayNone	1
xlIndicatorAndButton	0

### [xlSortDataOption](#)

Constant	Value
xlSortNormal	0
xlSortTextAsNumbers	1

### [xlSortMethod](#)

#### Constant Value

xlPinYin	1
xlStroke	2

### [xlSortMethodOld](#)

#### Constant Value

xlCodePage	2
xlSyllabary	1

## [xlSortOrder](#)

### **Constant Value**

xlAscending 1

xlDescending 2

## [xlSortOrientation](#)

### **Constant Value**

xlSortColumns 1

xlSortRows 2

## [xlSortType](#)

### **Constant Value**

xlSortLabels 2

xlSortValues 1

## [xlSourceType](#)

### **Constant Value**

xlSourceAutoFilter 3

xlSourceChart 5

xlSourcePivotTable 6

xlSourcePrintArea 2

xlSourceQuery 7

xlSourceRange 4

xlSourceSheet 1

xlSourceWorkbook 0

## [xlSpeakDirection](#)

### **Constant Value**

xlSpeakByColumns 1

xlSpeakByRows 0

### [xlSpecialCellsValue](#)

<b>Constant</b>	<b>Value</b>
xlErrors	16
xlLogical	4
xlNumbers	1
xlTextValues	2

### [xlSubscribeToFormat](#)

<b>Constant</b>	<b>Value</b>
xlSubscribeToPicture	-4147
xlSubscribeToText	-4158

### [xlSubtotalLocationType](#)

<b>Constant</b>	<b>Value</b>
xlAtBottom	2
xlAtTop	1

### [xlSummaryColumn](#)

<b>Constant</b>	<b>Value</b>
xlSummaryOnLeft	-4131
xlSummaryOnRight	-4152

### [xlSummaryReportType](#)

<b>Constant</b>	<b>Value</b>
xlStandardSummary	1
xlSummaryPivotTable	-4148

## [xlSummaryRow](#)

<b>Constant</b>	<b>Value</b>
xlSummaryAbove	0
xlSummaryBelow	1

## [xlTabPosition](#)

<b>Constant</b>	<b>Value</b>
xlTabPositionFirst	0
xlTabPositionLast	1

## [xlTextParsingType](#)

<b>Constant</b>	<b>Value</b>
xlDelimited	1
xlFixedWidth	2

## [xlTextQualifier](#)

<b>Constant</b>	<b>Value</b>
xlTextQualifierDoubleQuote	1
xlTextQualifierNone	-4142
xlTextQualifierSingleQuote	2

## [xlTextVisualLayoutType](#)

<b>Constant</b>	<b>Value</b>
xlTextVisualLTR	1
xlTextVisualRTL	2

## [xlTickLabelOrientation](#)

<b>Constant</b>	<b>Value</b>
-----------------	--------------

xlTickLabelOrientationAutomatic -4105  
xlTickLabelOrientationDownward -4170  
xlTickLabelOrientationHorizontal -4128  
xlTickLabelOrientationUpward -4171  
xlTickLabelOrientationVertical -4166

### [xlTickLabelPosition](#)

<b>Constant</b>	<b>Value</b>
xlTickLabelPositionHigh	-4127
xlTickLabelPositionLow	-4134
xlTickLabelPositionNextToAxis	4
xlTickLabelPositionNone	-4142

### [xlTickMark](#)

<b>Constant</b>	<b>Value</b>
xlTickMarkCross	4
xlTickMarkInside	2
xlTickMarkNone	-4142
xlTickMarkOutside	3

### [xlTimeUnit](#)

#### **Constant Value**

xlDays 0  
xlMonths 1  
xlYears 2

### [xlToolbarProtection](#)

<b>Constant</b>	<b>Value</b>
xlNoButtonChanges	1
xlNoChanges	4

xlNoDockingChanges 3  
xlNoShapeChanges 2  
xlToolbarProtectionNone -4143

### xlTotalsCalculation

<b>Constant</b>	<b>Value</b>
xlTotalsCalculationAverage	2
xlTotalsCalculationCount	3
xlTotalsCalculationCountNums	4
xlTotalsCalculationMax	6
xlTotalsCalculationMin	5
xlTotalsCalculationNone	0
xlTotalsCalculationStdDev	7
xlTotalsCalculationSum	1
xlTotalsCalculationVar	8

### xlTrendlineType

<b>Constant</b>	<b>Value</b>
xlExponential	5
xlLinear	-4132
xlLogarithmic	-4133
xlMovingAvg	6
xlPolynomial	3
xlPower	4

### xlUnderlineStyle

<b>Constant</b>	<b>Value</b>
xlUnderlineStyleDouble	-4119
xlUnderlineStyleDoubleAccounting	5
xlUnderlineStyleNone	-4142
xlUnderlineStyleSingle	2

xlUnderlineStyleSingleAccounting 4

### [xlUpdateLinks](#)

<b>Constant</b>	<b>Value</b>
xlUpdateLinksAlways	3
xlUpdateLinksNever	2
xlUpdateLinksUserSetting	1

### [xlVAlign](#)

<b>Constant</b>	<b>Value</b>
xlVAlignBottom	-4107
xlVAlignCenter	-4108
xlVAlignDistributed	-4117
xlVAlignJustify	-4130
xlVAlignTop	-4160

### [xlWBATemplate](#)

<b>Constant</b>	<b>Value</b>
xlWBATChart	-4109
xlWBATExcel4IntlMacroSheet	4
xlWBATExcel4MacroSheet	3
xlWBATWorksheet	-4167

### [xlWebFormatting](#)

<b>Constant</b>	<b>Value</b>
xlWebFormattingAll	1
xlWebFormattingNone	3
xlWebFormattingRTF	2

### [xlWebSelectionType](#)

<b>Constant</b>	<b>Value</b>
xlAllTables	2
xlEntirePage	1
xlSpecifiedTables	3

### [xlWindowState](#)

<b>Constant</b>	<b>Value</b>
xlMaximized	-4137
xlMinimized	-4140
xlNormal	-4143

### [xlWindowType](#)

<b>Constant</b>	<b>Value</b>
xlChartAsWindow	5
xlChartInPlace	4
xlClipboard	3
xlInfo	-4129
xlWorkbook	1

### [xlWindowView](#)

<b>Constant</b>	<b>Value</b>
xlNormalView	1
xlPageBreakPreview	2

### [xlXLMMacroType](#)

<b>Constant</b>	<b>Value</b>
xlCommand	2
xlFunction	1
xlNotXLM	3

## [xlXmlExportResult](#)

Constant	Value
xlXmlExportSuccess	0
xlXmlExportValidationFailed	1

## [xlXmlImportResult](#)

Constant	Value
xlXmlImportElementsTruncated	1
xlXmlImportSuccess	0
xlXmlImportValidationFailed	2

## [xlXmlLoadOption](#)

Constant	Value
xlXmlLoadImportToList	2
xlXmlLoadMapXml	3
xlXmlLoadOpenXml	1
xlXmlLoadPromptUser	0

## [xlYesNoGuess](#)

### Constant Value

xlGuess	0
xlNo	2
xlYes	1

---

# ChartObject Object Model

## ChartObject



- [Corners](#)
- [DataTable](#)
  - [Font](#)
- [Floor](#)
  - [ChartFillFormat](#)
- [Hyperlinks](#)
- [Legend](#)
  - [ChartFillFormat](#)
  - [Font](#)
- [PageSetup](#)
  - [Graphic](#)
- [PivotLayout](#)
  - [PivotTable](#)
- [PlotArea](#)
  - [ChartFillFormat](#)
- [Shapes](#)
- [Tab](#)
- [Walls](#)
  - [ChartFillFormat](#)

- [PivotItemList](#)
- [PivotField](#)
  - [CubeField](#)
- [PivotItem](#)
- [PivotTable](#)
  - [CalculatedMembers](#)
  - [CubeFields](#)
  - [PivotFormulas](#)
- [QueryTable](#)
  - [Parameters](#)
- [SoundNote](#)
- [Validation](#)
- [Worksheet](#)
  - [AutoFilter](#)
  - [Comments](#)
  - [CustomProperties](#)
  - [HPageBreaks](#)
  - [ListObjects](#)
  - [Names](#)
  - [Outline](#)
  - [PageSetup](#)
  - [Protection](#)
  - [QueryTables](#)
  - [Shapes](#)
  - [Tab](#)
  - [VPageBreaks](#)
- [XPath](#)
  - [XmlMap](#)

- [PictureFormat](#)
- [ShadowFormat](#)
  - [ColorFormat](#)
- [Shape](#)
  - [ControlForm:](#)
  - [LinkFormat](#)
  - [OLEFormat](#)
- [ShapeNodes](#)
- [TextEffectForm](#)
- [TextFrame](#)
- [ThreeDFormat](#)
  - [ColorFormat](#)

### **Legend**

Object and collecti  
Object only



# Name Object Model

<u>Name</u>	<u>Range</u>
└ <u>Areas</u>	
└ <u>Borders</u>	
└ <u>Border</u>	
└ <u>Characters</u>	
└ <u>Comment</u>	
└ <u>Shape</u>	
└ <u>Errors</u>	
└ <u>Error</u>	
└ <u>Font</u>	
└ <u>FormatConditions</u>	
└ <u>Hyperlinks</u>	
└ <u>Interior</u>	
└ <u>ListObject</u>	
└ <u>ListColumns</u>	
└ <u>ListRows</u>	
└ <u>XmlMap</u>	
└ <u>Phonetic</u>	
└ <u>Phonetics</u>	
└ <u>PivotCell</u>	
└ <u>PivotItemList</u>	
└ <u>PivotField</u>	
└ <u>CubeField</u>	
└ <u>PivotItem</u>	
└ <u>PivotTable</u>	
└ <u>CalculatedMembers</u>	



## Legend

Object and collection

Object only

# Shapes Collection Object Model

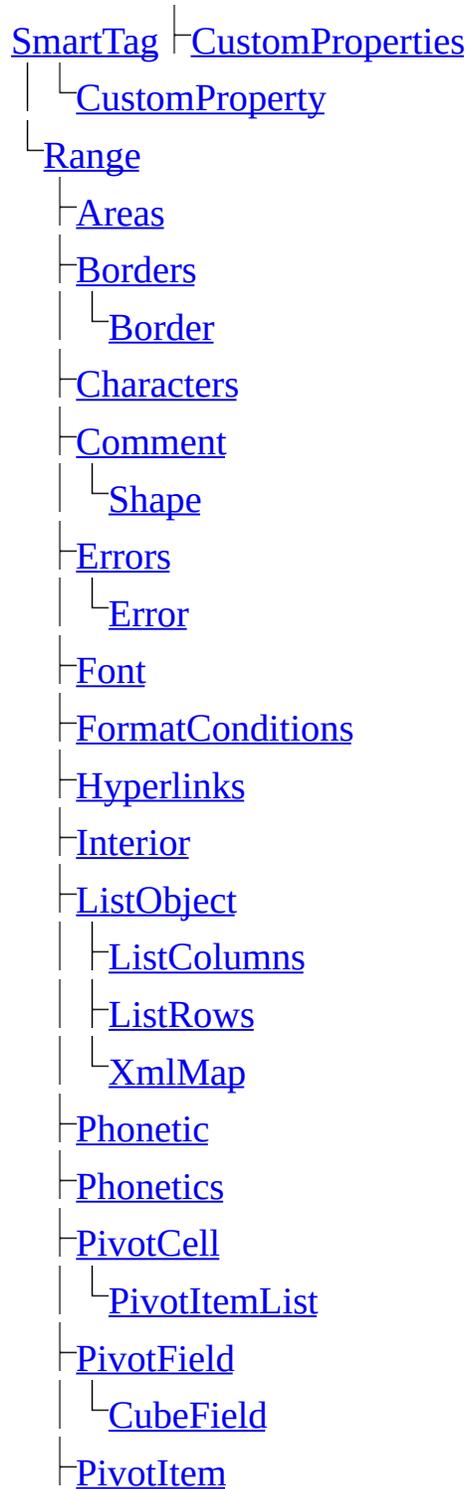


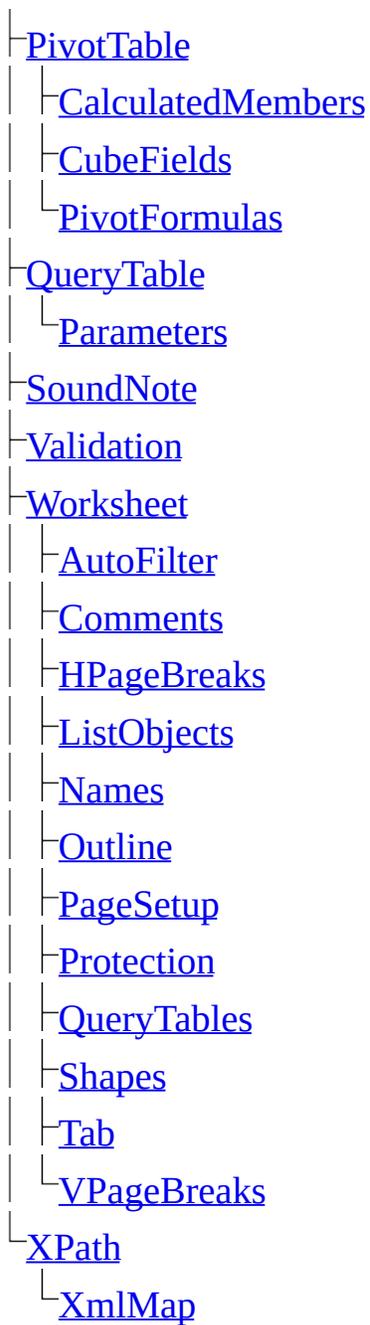
## Legend

Object and collection

Object only

# SmartTag Object Model





## Legend

Object and collection

Object only

# Built-In Dialog Box Argument Lists

<b>Dialog box constant</b>	<b>Argument list(s)</b>
xlDialogActivate	window_text, pane_num
xlDialogActiveCellFont	font, font_style, size, strikethrough, superscript, subscript, outline, shadow, underline, color, normal, background, start_char, char_count
xlDialogAddChartAutoformat	name_text, desc_text
xlDialogAddinManager	operation_num, addinname_text, copy_logical
xlDialogAlignment	horiz_align, wrap, vert_align, orientation, add_indent
xlDialogApplyNames	name_array, ignore, use_rowcol, omit_col, omit_row, order_num, append_last
xlDialogApplyStyle	style_text
xlDialogAppMove	x_num, y_num
xlDialogAppSize	x_num, y_num
xlDialogArrangeAll	arrange_num, active_doc, sync_horiz, sync_vert
xlDialogAssignToObject	macro_ref
xlDialogAssignToTool	bar_id, position, macro_ref
xlDialogAttachText	attach_to_num, series_num, point_num
xlDialogAttachToolbars	
xlDialogAutoCorrect	correct_initial_caps, capitalize_days
xlDialogAxes	x_primary, y_primary, x_secondary, y_secondary
xlDialogAxes	x_primary, y_primary, z_primary
xlDialogBorder	outline, left, right, top, bottom, shade, outline_color, left_color, right_color, top_color, bottom_color
xlDialogCalculation	type_num, iter, max_num, max_change, update, precision, date_1904, calc_save,

	save_values, alt_exp, alt_form
xlDialogCellProtection	locked, hidden
xlDialogChangeLink	old_text, new_text, type_of_link
xlDialogChartAddData	ref, rowcol, titles, categories, replace, series
xlDialogChartLocation	
xlDialogChartOptionsDataLabels	
xlDialogChartOptionsDataTable	
xlDialogChartSourceData	
xlDialogChartTrend	type, ord_per, forecast, backcast, intercept, equation, r_squared, name
xlDialogChartType	
xlDialogChartWizard	long, ref, gallery_num, type_num, plot_by, categories, ser_titles, legend, title, x_title, y_title, z_title, number_cats, number_titles
xlDialogCheckboxProperties	value, link, accel_text, accel2_text, 3d_shading
xlDialogClear	type_num
xlDialogColorPalette	file_text
xlDialogColumnWidth	width_num, reference, standard, type_num, standard_num
xlDialogCombination	type_num
xlDialogConditionalFormatting	
xlDialogConsolidate	source_refs, function_num, top_row, left_col, create_links
xlDialogCopyChart	size_num
xlDialogCopyPicture	appearance_num, size_num, type_num
xlDialogCreateNames	top, left, bottom, right
xlDialogCreatePublisher	file_text, appearance, size, formats
xlDialogCustomizeToolbar	category
xlDialogCustomViews	
xlDialogDataDelete	
xlDialogDataLabel	show_option, auto_text, show_key
xlDialogDataSeries	rowcol, type_num, date_num, step_value, stop_value, trend

xlDialogDataValidation	
xlDialogDefineName	name_text, refers_to, macro_type, shortcut_text, hidden, category, local
xlDialogDefineStyle	style_text, number, font, alignment, border, pattern, protection
xlDialogDefineStyle	style_text, attribute_num, additional_def_args, ...
xlDialogDeleteFormat	format_text
xlDialogDeleteName	name_text
xlDialogDemote	row_col
xlDialogDisplay	formulas, gridlines, headings, zeros, color_num, reserved, outline, page_breaks, object_num
xlDialogDisplay	cell, formula, value, format, protection, names, precedents, dependents, note
xlDialogEditboxProperties	validation_num, multiline_logical, vscroll_logical, password_logical
xlDialogEditColor	color_num, red_value, green_value, blue_value
xlDialogEditDelete	shift_num
xlDialogEditionOptions	edition_type, edition_name, reference, option, appearance, size, formats
xlDialogEditSeries	series_num, name_ref, x_ref, y_ref, z_ref, plot_order
xlDialogErrorbarX	include, type, amount, minus
xlDialogErrorbarY	include, type, amount, minus
xlDialogExternalDataProperties	
xlDialogExtract	unique
xlDialogFileDelete	file_text
xlDialogFileSharing	
xlDialogFillGroup	type_num
xlDialogFillWorkgroup	type_num
xlDialogFilter	
xlDialogFilterAdvanced	operation, list_ref, criteria_ref, copy_ref,

	unique
xlDialogFindFile	
xlDialogFont	name_text, size_num
xlDialogFontProperties	font, font_style, size, strikethrough, superscript, subscript, outline, shadow, underline, color, normal, background, start_char, char_count
xlDialogFormatAuto	format_num, number, font, alignment, border, pattern, width
xlDialogFormatChart	layer_num, view, overlap, angle, gap_width, gap_depth, chart_depth, doughnut_size, axis_num, drop, hilo, up_down, series_line, labels, vary
xlDialogFormatCharttype	apply_to, group_num, dimension, type_num
xlDialogFormatFont	color, backgd, apply, name_text, size_num, bold, italic, underline, strike, outline, shadow, object_id, start_num, char_num
xlDialogFormatFont	name_text, size_num, bold, italic, underline, strike, color, outline, shadow
xlDialogFormatFont	name_text, size_num, bold, italic, underline, strike, color, outline, shadow, object_id_text, start_num, char_num
xlDialogFormatLegend	position_num
xlDialogFormatMain	type_num, view, overlap, gap_width, vary, drop, hilo, angle, gap_depth, chart_depth, up_down, series_line, labels, doughnut_size
xlDialogFormatMove	x_offset, y_offset, reference
xlDialogFormatMove	x_pos, y_pos
xlDialogFormatMove	explosion_num
xlDialogFormatNumber	format_text
xlDialogFormatOverlay	type_num, view, overlap, gap_width, vary, drop, hilo, angle, series_dist, series_num, up_down, series_line, labels, doughnut_size
xlDialogFormatSize	width, height
xlDialogFormatSize	x_off, y_off, reference

xlDialogFormatText	x_align, y_align, orient_num, auto_text, auto_size, show_key, show_value, add_indent
xlDialogFormulaFind	text, in_num, at_num, by_num, dir_num, match_case, match_byte
xlDialogFormulaGoto	reference, corner
xlDialogFormulaReplace	find_text, replace_text, look_at, look_by, active_cell, match_case, match_byte
xlDialogFunctionWizard	
xlDialogGallery3dArea	type_num
xlDialogGallery3dBar	type_num
xlDialogGallery3dColumn	type_num
xlDialogGallery3dLine	type_num
xlDialogGallery3dPie	type_num
xlDialogGallery3dSurface	type_num
xlDialogGalleryArea	type_num, delete_overlay
xlDialogGalleryBar	type_num, delete_overlay
xlDialogGalleryColumn	type_num, delete_overlay
xlDialogGalleryCustom	name_text
xlDialogGalleryDoughnut	type_num, delete_overlay
xlDialogGalleryLine	type_num, delete_overlay
xlDialogGalleryPie	type_num, delete_overlay
xlDialogGalleryRadar	type_num, delete_overlay
xlDialogGalleryScatter	type_num, delete_overlay
xlDialogGoalSeek	target_cell, target_value, variable_cell
xlDialogGridlines	x_major, x_minor, y_major, y_minor, z_major, z_minor, 2D_effect
xlDialogImportTextFile	
xlDialogInsert	shift_num
xlDialogInsertHyperlink	
xlDialogInsertNameLabel	
xlDialogInsertObject	object_class, file_name, link_logical, display_icon_logical, icon_file, icon_number, icon_label
xlDialogInsertPicture	file_name, filter_number

xlDialogInsertTitle	chart, y_primary, x_primary, y_secondary, x_secondary
xlDialogLabelProperties	accel_text, accel2_text, 3d_shading
xlDialogListboxProperties	range, link, drop_size, multi_select, 3d_shading
xlDialogMacroOptions	macro_name, description, menu_on, menu_text, shortcut_on, shortcut_key, function_category, status_bar_text, help_id, help_file
xlDialogMailEditMailer	to_recipients, cc_recipients, bcc_recipients, subject, enclosures, which_address
xlDialogMailLogon	name_text, password_text, download_logical
xlDialogMailNextLetter	
xlDialogMainChart	type_num, stack, 100, vary, overlap, drop, hilo, overlap%, cluster, angle
xlDialogMainChartType	type_num
xlDialogMenuEditor	
xlDialogMove	x_pos, y_pos, window_text
xlDialogNew	type_num, xy_series, add_logical
xlDialogNewWebQuery	
xlDialogNote	add_text, cell_ref, start_char, num_chars
xlDialogObjectProperties	placement_type, print_object
xlDialogObjectProtection	locked, lock_text
xlDialogOpen	file_text, update_links, read_only, format, prot_pwd, write_res_pwd, ignore_rorec, file_origin, custom_delimit, add_logical, editable, file_access, notify_logical, converter
xlDialogOpenLinks	document_text1, document_text2, ..., read_only, type_of_link
xlDialogOpenMail	subject, comments
xlDialogOpenText	file_name, file_origin, start_row, file_type, text_qualifier, consecutive_delim, tab, semicolon, comma, space, other, other_char, field_info
	type_num, iter, max_num, max_change,

xlDialogOptionsCalculation	update, precision, date_1904, calc_save, save_values
xlDialogOptionsChart	display_blanks, plot_visible, size_with_window
xlDialogOptionsEdit	incell_edit, drag_drop, alert, entermove, fixed, decimals, copy_objects, update_links, move_direction, autocomplete, animations
xlDialogOptionsGeneral	R1C1_mode, dde_on, sum_info, tips, recent_files, old_menus, user_info, font_name, font_size, default_location, alternate_location, sheet_num, enable_under
xlDialogOptionsListsAdd	string_array
xlDialogOptionsListsAdd	import_ref, by_row
xlDialogOptionsME	def_rtl_sheet, crsr_mvmt, show_ctrl_char, gui_lang
xlDialogOptionsTransition	menu_key, menu_key_action, nav_keys, trans_eval, trans_entry
xlDialogOptionsView	formula, status, notes, show_info, object_num, page_breaks, formulas, gridlines, color_num, headers, outline, zeros, hor_scroll, vert_scroll, sheet_tabs
xlDialogOutline	auto_styles, row_dir, col_dir, create_apply
xlDialogOverlay	type_num, stack, 100, vary, overlap, drop, hilo, overlap%, cluster, angle, series_num, auto
xlDialogOverlayChartType	type_num
xlDialogPageSetup	head, foot, left, right, top, bot, hdng, grid, h_cntr, v_cntr, orient, paper_size, scale, pg_num, pg_order, bw_cells, quality, head_margin, foot_margin, notes, draft
xlDialogPageSetup	head, foot, left, right, top, bot, size, h_cntr, v_cntr, orient, paper_size, scale, pg_num, bw_chart, quality, head_margin, foot_margin, draft
xlDialogPageSetup	head, foot, left, right, top, bot, orient, paper_size, scale, quality, head_margin,

	foot_margin, pg_num
xlDialogParse	parse_text, destination_ref
xlDialogPasteNames	
xlDialogPasteSpecial	paste_num, operation_num, skip_blanks, transpose
xlDialogPasteSpecial	rowcol, titles, categories, replace, series
xlDialogPasteSpecial	paste_num
xlDialogPasteSpecial	format_text, pastelink_logical, display_icon_logical, icon_file, icon_number, icon_label
xlDialogPatterns	apattern, afore, aback, newui
xlDialogPatterns	lauto, lstyle, lcolor, lwt, hwidth, hlength, htype
xlDialogPatterns	bauto, bstyle, bcolor, bwt, shadow, aauto, apattern, afore, aback, rounded, newui
xlDialogPatterns	bauto, bstyle, bcolor, bwt, shadow, aauto, apattern, afore, aback, invert, apply, newfill
xlDialogPatterns	lauto, lstyle, lcolor, lwt, tmajor, tminor, tlabel
xlDialogPatterns	lauto, lstyle, lcolor, lwt, apply, smooth
xlDialogPatterns	lauto, lstyle, lcolor, lwt, mauto, mstyle, mfore, mback, apply, smooth
xlDialogPatterns	type, picture_units, apply
xlDialogPhonetic	
xlDialogPivotCalculatedField	
xlDialogPivotCalculatedItem	
xlDialogPivotClientServerSet	
xlDialogPivotFieldGroup	start, end, by, periods
xlDialogPivotFieldProperties	name, pivot_field_name, new_name, orientation, function, formats
xlDialogPivotFieldUngroup	
xlDialogPivotShowPages	name, page_field
xlDialogPivotSolveOrder	
xlDialogPivotTableOptions	type, source, destination, name, row_grand,

xlDialogPivotTableWizard	col_grand, save_data, apply_auto_format, auto_page, reserved
xlDialogPlacement	placement_type
xlDialogPrint	range_num, from, to, copies, draft, preview, print_what, color, feed, quality, y_resolution, selection, printer_text, print_to_file, collate
xlDialogPrinterSetup	printer_text
xlDialogPrintPreview	
xlDialogPromote	rowcol
xlDialogProperties	title, subject, author, keywords, comments
xlDialogProtectDocument	contents, windows, password, objects, scenarios
xlDialogProtectSharing	
xlDialogPublishAsWebPage	
xlDialogPushbuttonProperties	default_logical, cancel_logical, dismiss_logical, help_logical, accel_text, accel_text2
xlDialogReplaceFont	font_num, name_text, size_num, bold, italic, underline, strike, color, outline, shadow
xlDialogRoutingSlip	recipients, subject, message, route_num, return_logical, status_logical
xlDialogRowHeight	height_num, reference, standard_height, type_num
xlDialogRun	reference, step
xlDialogSaveAs	document_text, type_num, prot_pwd, backup, write_res_pwd, read_only_rec
xlDialogSaveCopyAs	document_text
xlDialogSaveNewObject	
xlDialogSaveWorkbook	document_text, type_num, prot_pwd, backup, write_res_pwd, read_only_rec
xlDialogSaveWorkspace	name_text
xlDialogScale	cross, cat_labels, cat_marks, between, max, reverse
xlDialogScale	min_num, max_num, major, minor, cross, logarithmic, reverse, max

xlDialogScale	cat_labels, cat_marks, reverse, between
xlDialogScale	series_labels, series_marks, reverse
xlDialogScale	min_num, max_num, major, minor, cross, logarithmic, reverse, min
xlDialogScenarioAdd	scen_name, value_array, changing_ref, scen_comment, locked, hidden
xlDialogScenarioCells	changing_ref
xlDialogScenarioEdit	scen_name, new_scenname, value_array, changing_ref, scen_comment, locked, hidden
xlDialogScenarioMerge	source_file
xlDialogScenarioSummary	result_ref, report_type
xlDialogScrollbarProperties	value, min, max, inc, page, link, 3d_shading
xlDialogSelectSpecial	type_num, value_type, levels
xlDialogSendMail	recipients, subject, return_receipt
xlDialogSeriesAxes	axis_num
xlDialogSeriesOptions	
xlDialogSeriesOrder	chart_num, old_series_num, new_series_num
xlDialogSeriesShape	
xlDialogSeriesX	x_ref
xlDialogSeriesY	name_ref, y_ref
xlDialogSetBackgroundPicture	
xlDialogSetPrintTitles	titles_for_cols_ref, titles_for_rows_ref
xlDialogSetUpdateStatus	link_text, status, type_of_link
xlDialogShowDetail	rowcol, rowcol_num, expand, show_field
xlDialogShowToolbar	bar_id, visible, dock, x_pos, y_pos, width, protect, tool_tips, large_buttons, color_buttons
xlDialogSize	width, height, window_text
xlDialogSort	orientation, key1, order1, key2, order2, key3, order3, header, custom, case
xlDialogSort	orientation, key1, order1, type, custom
xlDialogSortSpecial	sort_by, method, key1, order1, key2, order2, key3, order3, header, order, case
xlDialogSplit	col_split, row_split

xlDialogStandardFont	name_text, size_num, bold, italic, underline, strike, color, outline, shadow
xlDialogStandardWidth	standard_num
xlDialogStyle	bold, italic
xlDialogSubscribeTo	file_text, format_num
xlDialogSubtotalCreate	at_change_in, function_num, total, replace, pagebreaks, summary_below
xlDialogSummaryInfo	title, subject, author, keywords, comments
xlDialogTable	row_ref, column_ref
xlDialogTabOrder	
xlDialogTextToColumns	destination_ref, data_type, text_delim, consecutive_delim, tab, semicolon, comma, space, other, other_char, field_info
xlDialogUnhide	window_text
xlDialogUpdateLink	link_text, type_of_link
xlDialogVbaInsertFile	filename_text
xlDialogVbaMakeAddIn	
xlDialogVbaProcedureDefinition	
xlDialogView3d	elevation, perspective, rotation, axes, height%, autoscale
xlDialogWebOptionsEncoding	
xlDialogWebOptionsFiles	
xlDialogWebOptionsFonts	
xlDialogWebOptionsGeneral	
xlDialogWebOptionsPictures	
xlDialogWindowMove	x_pos, y_pos, window_text
xlDialogWindowSize	width, height, window_text
xlDialogWorkbookAdd	name_array, dest_book, position_num
xlDialogWorkbookCopy	name_array, dest_book, position_num
xlDialogWorkbookInsert	type_num
xlDialogWorkbookMove	name_array, dest_book, position_num
xlDialogWorkbookName	oldname_text, newname_text
xlDialogWorkbookNew	
xlDialogWorkbookOptions	sheet_name, bound_logical, new_name

xlDialogWorkbookProtect	structure, windows, password
xlDialogWorkbookTabSplit	ratio_num
xlDialogWorkbookUnhide	sheet_text
xlDialogWorkgroup	name_array
	fixed, decimals, r1c1, scroll, status, formula,
xlDialogWorkspace	menu_key, remote, entermove, underlines,
	tools, notes, nav_keys, menu_key_action,
	drag_drop, show_info
xlDialogZoom	magnification



# Using ActiveX Controls on Sheets

This topic covers specific information about using ActiveX controls on worksheets and chart sheets. For general information on adding and working with controls, see [Using ActiveX Controls on a Document](#) and [Creating a Custom Dialog box](#).

Keep the following points in mind when you are working with controls on sheets.

- In addition to the standard properties available for ActiveX controls, the following properties can be used with ActiveX controls in Microsoft Excel: [BottomRightCell](#), [LinkedCell](#), [ListFillRange](#), [Placement](#), [PrintObject](#), [TopLeftCell](#), and [ZOrder](#).

These properties can be set and returned using the ActiveX control name. The following example scrolls the workbook window so CommandButton1 is in the upper-left corner.

```
Set t = Sheet1.CommandButton1.TopLeftCell
With ActiveWindow
    .ScrollRow = t.Row
    .ScrollColumn = t.Column
End With
```

- Some Microsoft Excel Visual Basic methods and properties are disabled when an ActiveX control is activated. For example, the **Sort** method cannot be used when a control is active, so the following code fails in a button click event procedure (because the control is still active after the user clicks it).

```
Private Sub CommandButton1.Click
    Range("a1:a10").Sort Key1:=Range("a1")
End Sub
```

You can work around this problem by activating some other element on the sheet before you use the property or method that failed. For example, the following

code sorts the range:

```
Private Sub CommandButton1.Click
    Range("a1").Activate
    Range("a1:a10").Sort Key1:=Range("a1")
    CommandButton1.Activate
End Sub
```

- Controls on a Microsoft Excel workbook embedded in a document in another application will not work if the user double clicks the workbook to edit it. The controls will work if the user right clicks the workbook and selects the **Open** command from the shortcut menu.
- When a Microsoft Excel workbook is saved using the Microsoft Excel 5.0/95 Workbook file format, ActiveX control information is lost.
- The **Me** keyword in an event procedure for an ActiveX control on a sheet refers to the sheet, not to the control.

## Adding Controls with Visual Basic

In Microsoft Excel, ActiveX controls are represented by **OLEObject** objects in the **OLEObjects** collection (all **OLEObject** objects are also in the **Shapes** collection). To programmatically add an ActiveX control to a sheet, use the **Add** method of the **OLEObjects** collection. The following example adds a command button to worksheet one.

```
Worksheets(1).OLEObjects.Add "Forms.CommandButton.1", _  
    Left:=10, Top:=10, Height:=20, Width:=100
```

## Using Control Properties with Visual Basic

Most often, your Visual Basic code will refer to ActiveX controls by name. The following example changes the caption on the control named "CommandButton1."

```
Sheet1.CommandButton1.Caption = "Run"
```

Note that when you use a control name outside the class module for the sheet containing the control, you must qualify the control name with the sheet name.

To change the control name you use in Visual Basic code, select the control and set the **(Name)** property in the Properties window.

Because ActiveX controls are also represented by **OLEObject** objects in the **OLEObjects** collection, you can set control properties using the objects in the collection. The following example sets the left position of the control named "CommandButton1."

```
Worksheets(1).OLEObjects("CommandButton1").Left = 10
```

Control properties that are not shown as properties of the **OLEObject** object can be set by returning the actual control object using the **Object** property. The following example sets the caption for CommandButton1.

```
Worksheets(1).OLEObjects("CommandButton1"). _  
    Object.Caption = "run me"
```

Because all OLE objects are also members of the **Shapes** collection, you can use the collection to set properties for several controls. The following example aligns the left edge of all controls on worksheet one.

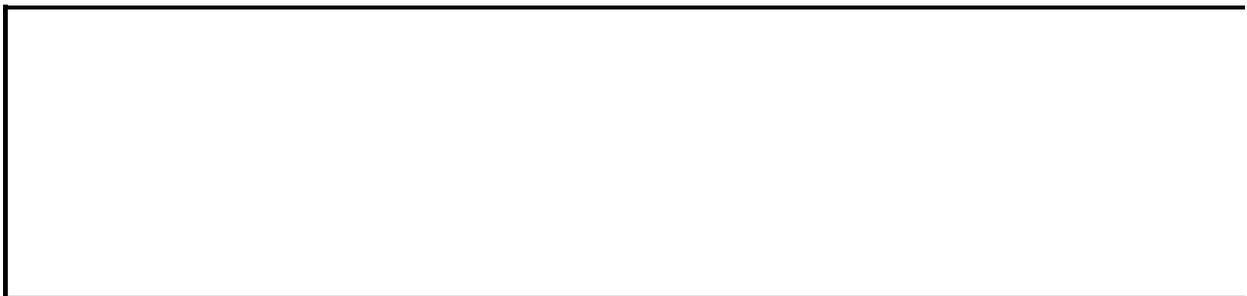
```
For Each s In Worksheets(1).Shapes  
    If s.Type = msoOLEControlObject Then s.Left = 10  
Next
```

## Using Control Names with the Shapes and OLEObjects Collections

An ActiveX control on a sheet has two names: the name of the shape that contains the control, which you can see in the **Name** box when you view the sheet, and the code name for the control, which you can see in the cell to the right of **(Name)** in the Properties window. When you first add a control to a sheet, the shape name and code name match. However, if you change either the shape name or code name, the other isn't automatically changed to match.

You use the code name of a control in the names of its event procedures. However, when you return a control from the **Shapes** or **OLEObjects** collection for a sheet, you must use the shape name, not the code name, to refer to the control by name. For example, assume that you add a check box to a sheet and that both the default shape name and the default code name are `CheckBox1`. If you then change the control code name by typing **chkFinished** next to **(Name)** in the Properties window, you must use `chkFinished` in event procedures names, but you still have to use `CheckBox1` to return the control from the **Shapes** or **OLEObject** collection, as shown in the following example.

```
Private Sub chkFinished_Click()  
    ActiveSheet.OLEObjects("CheckBox1").Object.Value = 1  
End Sub
```



# Working with Shapes (Drawing Objects)

Shapes, or drawing objects, are represented by three different objects: the [Shapes](#) collection, the [ShapeRange](#) collection, and the [Shape](#) object. In general, you use the **Shapes** collection to create shapes and to iterate through all the shapes on a given worksheet; you use the **Shape** object to format or modify a single shape; and you use the **ShapeRange** collection to modify multiple shapes the same way you work with multiple shapes in the user interface.

## Setting Properties for a Shape

Many formatting properties of shapes aren't set by properties that apply directly to the **Shape** or **ShapeRange** object. Instead, related shape attributes are grouped under secondary objects, such as the **FillFormat** object, which contains all the properties that relate to the shape's fill, or the **LinkFormat** object, which contains all the properties that are unique to linked OLE objects. To set properties for a shape, you must first return the object that represents the set of related shape attributes and then set properties of that returned object. For example, you use the **Fill** property to return the **FillFormat** object, and then you set the **ForeColor** property of the **FillFormat** object to set the fill foreground color for the specified shape, as shown in the following example.

```
Worksheets(1).Shapes(1).Fill.ForeColor.RGB = RGB(255, 0, 0)
```

## Applying a Property or Method to Several Shapes at the Same Time

In the user interface, there are some operations you can perform with several shapes selected; for example, you can select several shapes and set all their individual fills at once. There are other operations you can only perform with a single shape selected; for example, you can only edit the text in a shape if a single shape is selected.

In Visual Basic, there are two ways to apply properties and methods to a set of shapes. These two ways allow you to perform any operation that you can perform on a single shape on a range of shapes, whether or not you can perform the same operation in the user interface.

- If the operation works on a multiple selected shapes in the user interface, you can perform the same operation in Visual Basic by constructing a **ShapeRange** collection that contains the shapes you want to work with, and applying the appropriate properties and methods directly to the **ShapeRange** collection.
- If the operation doesn't work on multiple selected shapes in the user interface, you can still perform the operation in Visual Basic by looping through the **Shapes** collection or through a **ShapeRange** collection that contains the shapes you want to work with, and applying the appropriate properties and methods to the individual **Shape** objects in the collection.

Many properties and methods that apply to the **Shape** object and **ShapeRange** collection fail if applied to certain kinds of shapes. For example, the **TextFrame** property fails if applied to a shape that cannot contain text. If you are not positive that each the shapes in a **ShapeRange** collection can have a certain property or method applied to it, don't apply the property or method to the **ShapeRange** collection. If you want to apply one of these properties or methods to a collection of shapes, you must loop through the collection and test each individual shape to make sure it's an appropriate type of shape before applying to property or method to it.

## Creating a ShapeRange Collection that Contains All Shapes on a Sheet

You can create a **ShapeRange** object that contains all the **Shape** objects on a sheet by selecting the shapes and then using the **ShapeRange** property to return a **ShapeRange** object containing the selected shapes.

```
Worksheets(1).Shapes.Select  
Set sr = Selection.ShapeRange
```

In Microsoft Excel, the **Index** argument isn't optional for the **Range** property of the **Shapes** collection, so you cannot use this property without an argument to create a **ShapeRange** object containing all shapes in a **Shapes** collection.

## Applying a Property or Method to a ShapeRange Collection

If you can perform an operation on multiple selected shapes in the user interface at the same time, you can do the programmatic equivalent by constructing a **ShapeRange** collection and then applying the appropriate properties or methods to it. The following example constructs a shape range that contains the shapes named "Big Star" and "Little Star" on myDocument and applies a gradient fill to them.

```
Set myDocument = Worksheets(1)
Set myRange = myDocument.Shapes.Range(Array("Big Star", _
    "Little Star"))
myRange.Fill.PresetGradient _
    msoGradientHorizontal, 1, msoGradientBrass
```

The following are general guidelines for how properties and methods behave when they're applied to a **ShapeRange** collection.

- Applying a method to the collection is equivalent to applying the method to each individual **Shape** object in that collection.
- Setting the value of a property of the collection is equivalent to setting the value of the property of each individual shape in that range.
- A property of the collection that returns a constant returns the value of the property for an individual shape in the collection if all shapes in the collection have the same value for that property. If not all shapes in the collection have the same value for the property, it returns the "mixed" constant.
- A property of the collection that returns a simple data type (such as **Long**, **Single**, or **String**) returns the value of the property for an individual shape if all shapes in the collection have the same value for that property.
- The value of some properties can be returned or set only if there's exactly one shape in the collection. If there's more than one shape in the collection, a run-time error occurs. This is generally the case for returning or setting properties when the equivalent action in the user interface is possible only with a single shape (actions such as editing text in a shape or editing the points of a freeform).

The preceding guidelines also apply when you are setting properties of shapes that are grouped under secondary objects of the **ShapeRange** collection, such as the **FillFormat** object. If the secondary object represents operations that can be performed on multiple selected objects in the user interface, you will be able to return the object from a **ShapeRange** collection and set its properties. For example, you can use the **Fill** property to return the **FillFormat** object that represents the fills of all the shapes in the **ShapeRange** collection. Setting the properties of this **FillFormat** object will set the same properties for all the individual shapes in the **ShapeRange** collection.

## Looping Through a Shapes or ShapeRange Collection

Even if you cannot perform an operation on several shapes in the user interface at the same time by selecting them and then using a command, you can perform the equivalent action programmatically by looping through a **Shapes** or **ShapeRange** collection that contains the shapes you want to work with, applying the appropriate properties and methods to the individual **Shape** objects in the collection. The following example loops through all the shapes on myDocument and changes the foreground color for each shape that's an AutoShape.

```
Set myDocument = Worksheets(1)
For Each sh In myDocument.Shapes
    If sh.Type = msoAutoShape Then
        sh.Fill.ForeColor.RGB = RGB(255, 0, 0)
    End If
Next
```

The following example constructs a **ShapeRange** collection that contains all the currently selected shapes in the active window and sets the foreground color for each selected shape.

```
For Each sh in ActiveWindow.Selection.ShapeRange
    sh.Fill.ForeColor.RGB = RGB(255, 0, 0)
Next
```

## Aligning, Distributing, and Grouping Shapes in a Shape Range

Use the [Align](#) and [Distribute](#) methods to position a set of shapes relative to one another or relative to the document that contains them. Use the [Group](#) method or the [Regroup](#) method to form a single grouped shape from a set of shapes.



# Group Method (ShapeRange Object)

Groups the shapes in the specified range. Returns the grouped shapes as a single **Shape** object.

*expression*.**Group**

*expression* Required. An expression that returns a **ShapeRange** object.

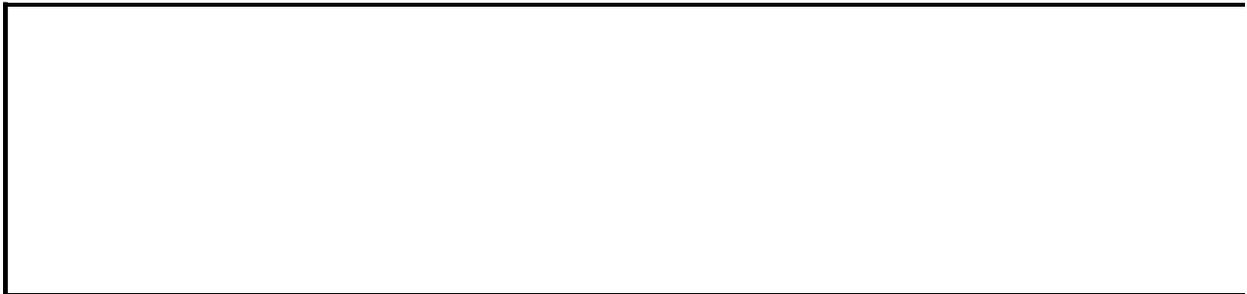
## Remarks

Because a group of shapes is treated as a single shape, grouping and ungrouping shapes changes the number of items in the **Shapes** collection and changes the index numbers of items that come after the affected items in the collection.

## Example

This example adds two shapes to myDocument, groups the two new shapes, sets the fill for the group, rotates the group, and sends the group to the back of the drawing layer.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes
    .AddShape(msoShapeCan, 50, 10, 100, 200).Name = "shpOne"
    .AddShape(msoShapeCube, 150, 250, 100, 200).Name = "shpTwo"
    With .Range(Array("shpOne", "shpTwo")).Group
        .Fill.PresetTextured msoTextureBlueTissuePaper
        .Rotation = 45
        .ZOrder msoSendToBack
    End With
End With
```



# AddOLEObject Method

Creates an OLE object. Returns a [Shape](#) object that represents the new OLE object.

*expression*.AddOLEObject(*ClassType*, *FileName*, *Link*, *DisplayAsIcon*, *IconFileName*, *IconIndex*, *IconLabel*, *Left*, *Top*, *Width*, *Height*)

*expression* Required. An expression that returns a **Shapes** object.

**ClassType** Optional **Variant**. (you must specify either **ClassType** or **FileName**). A string that contains the programmatic identifier for the object to be created. If **ClassType** is specified, **FileName** and **Link** are ignored. For more information about programmatic identifiers, see [OLE Programmatic Identifiers](#).

**FileName** Optional **Variant**. The file from which the object is to be created. If the path isn't specified, the current working folder is used. You must specify either the **ClassType** or **FileName** argument for the object, but not both.

**Link** Optional **Variant**. **True** to link the OLE object to the file from which it was created. **False** to make the OLE object an independent copy of the file. If you specified a value for **ClassType**, this argument must be **False**. The default value is **False**.

**DisplayAsIcon** Optional **Variant**. **True** to display the OLE object as an icon. The default value is **False**.

**IconFileName** Optional **Variant**. The file that contains the icon to be displayed.

**IconIndex** Optional **Variant**. The index of the icon within **IconFileName**. The order of icons in the specified file corresponds to the order in which the icons appear in the **Change Icon** dialog box (accessed from the **Object** dialog box when the **Display as icon** check box is selected). The first icon in the file has the index number 0 (zero). If an icon with the given index number doesn't exist in **IconFileName**, the icon with the index number 1 (the second icon in the file) is used. The default value is 0 (zero).

**IconLabel** Optional **Variant**. A label (caption) to be displayed beneath the icon.

**Left** , **Top** Optional **Variant**. The position (in points) of the upper-left corner of the new object relative to the upper-left corner of the document. The default value is 0 (zero).

**Width** , **Height** Optional **Variant**. The initial dimensions of the OLE object, in points.

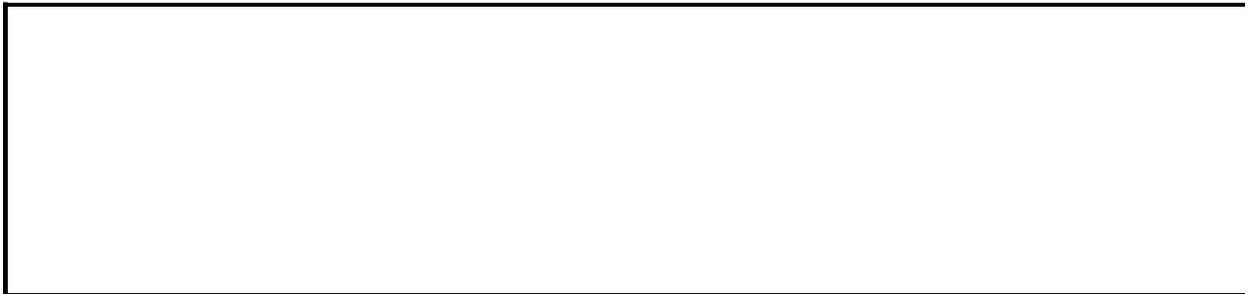
## Example

This example adds a linked Word document to myDocument.

```
Set myDocument = Worksheets(1)
myDocument.Shapes.AddOLEObject Left:=100, Top:=100, _
    Width:=200, Height:=300, _
    FileName:="c:\my documents\testing.doc", link:=True
```

This example adds a new command button to myDocument.

```
Set myDocument = Worksheets(1)
myDocument.Shapes.AddOLEObject Left:=100, Top:=100, _
    Width:=100, Height:=200, _
    ClassType:="Forms.CommandButton.1"
```



# Using Microsoft Excel Worksheet Functions in Visual Basic

You can use most Microsoft Excel worksheet functions in your Visual Basic statements. To see a list of the worksheet functions you can use, see [List of Worksheet Functions Available to Visual Basic](#).

**Note** Some worksheet functions aren't useful in Visual Basic. For example, the **Concatenate** function isn't needed because in Visual Basic you can use the **&** operator to join multiple text values.

## Calling a Worksheet Function from Visual Basic

In Visual Basic, the Microsoft Excel worksheet functions are available through the **WorksheetFunction** object.

The following **Sub** procedure uses the **Min** worksheet function to determine the smallest value in a range of cells. First, the variable `myRange` is declared as a **Range** object, and then it's set to range `A1:C10` on `Sheet1`. Another variable, `answer`, is assigned the result of applying the **Min** function to `myRange`. Finally, the value of `answer` is displayed in a message box.

```
Sub UseFunction()  
    Dim myRange As Range  
    Set myRange = Worksheets("Sheet1").Range("A1:C10")  
    answer = Application.WorksheetFunction.Min(myRange)  
    MsgBox answer  
End Sub
```

If you use a worksheet function that requires a range reference as an argument, you must specify a **Range** object. For example, you can use the **Match** worksheet function to search a range of cells. In a worksheet cell, you would enter a formula such as `=MATCH(9,A1:A10,0)`. However, in a Visual Basic procedure, you would specify a **Range** object to get the same result.

```
Sub FindFirst()  
    myVar = Application.WorksheetFunction _  
        .Match(9, Worksheets(1).Range("A1:A10"), 0)  
    MsgBox myVar  
End Sub
```

**Note** Visual Basic functions don't use the **WorksheetFunction** qualifier. A function may have the same name as a Microsoft Excel function and yet work differently. For example, `Application.WorksheetFunction.Log` and `Log` will return different values.

## Inserting a Worksheet Function into a Cell

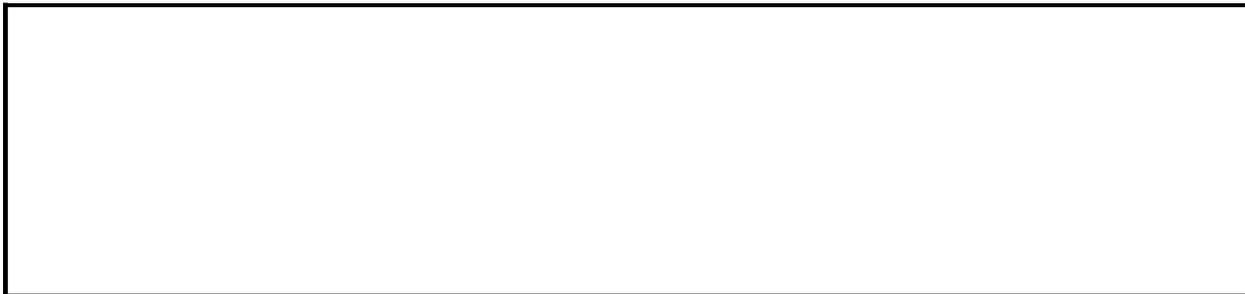
To insert a worksheet function into a cell, you specify the function as the value of the **Formula** property of the corresponding **Range** object. In the following example, the RAND worksheet function (which generates a random number) is assigned to the **Formula** property of range A1:B3 on Sheet1 in the active workbook.

```
Sub InsertFormula()  
    Worksheets("Sheet1").Range("A1:B3").Formula = "=RAND()"  
End Sub
```

## Example

This example uses the worksheet function **Pmt** to calculate a home mortgage loan payment. Notice that this example uses the **InputBox** method instead of the **InputBox** function so that the method can perform type checking. The **Static** statements cause Visual Basic to retain the values of the three variables; these are displayed as default values the next time you run the program.

```
Static loanAmt
Static loanInt
Static loanTerm
loanAmt = Application.InputBox _
    (Prompt:="Loan amount (100,000 for example)", _
    Default:=loanAmt, Type:=1)
loanInt = Application.InputBox _
    (Prompt:="Annual interest rate (8.75 for example)", _
    Default:=loanInt, Type:=1)
loanTerm = Application.InputBox _
    (Prompt:="Term in years (30 for example)", _
    Default:=loanTerm, Type:=1)
payment = Application.WorksheetFunction _
    .Pmt(loanInt / 1200, loanTerm * 12, loanAmt)
MsgBox "Monthly payment is " & Format(payment, "Currency")
```



[Show All](#)

# Modify Method (FormatCondition Object)

Modifies an existing conditional format.

*expression*.**Modify**(*Type*, *Operator*, *Formula1*, *Formula2*)

*expression* Required. An expression that returns a **FormatCondition** object.

*Type* Required [XlFormatCondition](#). Specifies whether the conditional format is based on a cell value or an expression.

XlFormatCondition can be one of these XlFormatCondition constants.

**xlCellValue**

**xlExpression**

*Operator* Optional [XlFormatConditionOperator](#). The conditional format operator.

XlFormatConditionOperator can be one of these XlFormatConditionOperator constants.

**xlBetween**

**xlEqual**

**xlGreater**

**xlGreaterEqual**

**xlLess**

**xlLessEqual**

**xlNotBetween**

**xlNotEqual**

If *Type*, is **xlExpression**, the *Operator* argument is ignored.

**Formula1** Optional **Variant**. The value or expression associated with the conditional format. Can be a constant value, a string value, a cell reference, or a formula.

**Formula2** Optional **Variant**. The value or expression associated with the conditional format. Can be a constant value, a string value, a cell reference, or a formula..

## Example

This example modifies an existing conditional format for cells E1:E10.

```
Worksheets(1).Range("e1:e10").FormatConditions(1) _  
    .Modify xlCellValue, xlLess, "=$a$1"
```



[Show All](#)

# Modify Method (Validation Object)

Modifies data validation for a range.

*expression*.**Modify**(*Type*, *AlertStyle*, *Operator*, *Formula1*, *Formula2*)

*expression* Required. An expression that returns a **Validation** object.

**Type** Required [XIDVType](#). The validation type.

XIDVType can be one of these XIDVType constants.

**xlValidateCustom**

**xlValidateDate**

**xlValidateDecimal**

**xlValidateInputOnly**

**xlValidateList**

**xlValidateTextLength**

**xlValidateTime**

**xlValidateWholeNumber**

**AlertStyle** Optional [XIDVAlertStyle](#). The validation alert style.

XIDVAlertStyle can be one of these XIDVAlertStyle constants.

**xlValidAlertInformation**

**xlValidAlertStop**

**xlValidAlertWarning**

**Operator** Optional [XIFormatConditionOperator](#). The data validation

operator.

XlFormatConditionOperator can be one of these XlFormatConditionOperator constants.

**xlBetween**

**xlEqual**

**xlGreater**

**xlGreaterEqual**

**xlLess**

**xlLessEqual**

**xlNotBetween**

**xlNotEqual**

**Formula1** Optional **Variant**. The first part of the data validation equation.

**Formula2** Optional **Variant**. The second part of the data validation when **Operator** is **xlBetween** or **xlNotBetween** (otherwise, this argument is ignored).

## Remarks

The **Modify** method requires different arguments, depending on the validation type, as shown in the following [table](#).

Validation type	Arguments
<b>xlInputOnly</b>	<i>AlertStyle</i> , <i>Formula1</i> , and <i>Formula2</i> are not used. <i>Formula1</i> is required; <i>Formula2</i> is ignored.
<b>xlValidateCustom</b>	<i>Formula1</i> must contain an expression that evaluates to <b>True</b> when data entry is valid and <b>False</b> when data entry is invalid. <i>Formula1</i> is required; <i>Formula2</i> is ignored.
<b>xlValidateList</b>	<i>Formula1</i> must contain either a comma-delimited list of values or a worksheet reference to the list.
<b>xlValidateDate,</b> <b>xlValidateDecimal,</b> <b>xlValidateTextLength,</b> <b>xlValidateTime,</b> or <b>xlValidateWholeNumber</b>	<i>Formula1</i> or <i>Formula2</i> , or both, must be specified.

## Example

This example changes data validation for cell E5.

```
Range("e5").Validation _  
    .Modify xlValidateList, xlValidAlertStop, _  
    xlBetween, "=$A$1:$A$10"
```



# FileName Property

Returns or sets the URL (on the intranet or the Web) or path (local or network) to the location where the specified source object was saved. Read/write **String**.

## Remarks

The **FileName** property generates an error if a folder in the specified path doesn't exist.

## Example

This example sets the location where the first item in the active workbook is to be saved.

```
ActiveWorkbook.PublishObjects(1).FileName = _  
    "\\Server2\Q1\StockReport.htm"
```



# ZOrderPosition Property

Returns the position of the specified shape in the z-order. Read-only **Long**.

## Remarks

To set the shape's position in the z-order, use the [ZOrder](#) method.

A shape's position in the z-order corresponds to the shape's index number in the **Shapes** collection. For example, if there are four shapes on `myDocument`, the expression `myDocument.Shapes(1)` returns the shape at the back of the z-order, and the expression `myDocument.Shapes(4)` returns the shape at the front of the z-order.

Whenever you add a new shape to a collection, it's added to the front of the z-order by default.

## Example

This example adds an oval to myDocument and then places the oval second from the back in the z-order if there is at least one other shape on the document.

```
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddShape(msoShapeOval, 100, 100, 100, 300)
    While .ZOrderPosition > 2
        .ZOrder msoSendBackward
    Wend
End With
```



[Show All](#)

# Returning an Object from a Collection

The **Item** property returns a single object from a collection. The following example sets the `firstBook` variable to a [Workbook](#) object that represents workbook one.

```
Set FirstBook = Workbooks.Item(1)
```

The **Item** property is the [default property](#) for most collections, so you can write the same statement more concisely by omitting the **Item** keyword.

```
Set FirstBook = Workbooks(1)
```

For more information about a specific collection, see the Help topic for that collection or the **Item** property for the collection.

## Named Objects

Although you can usually specify an integer value with the **Item** property, it may be more convenient to return an object by name. Before you can use a name with the **Item** property, you must name the object. Most often, this is done by setting the object's **Name** property. The following example creates a named worksheet in the active workbook and then refers to the worksheet by name.

```
ActiveWorkbook.Worksheets.Add.Name = "A New Sheet"  
With Worksheets("A New Sheet")  
    .Range("A5:A10").Formula = "=RAND()"  
End With
```

## Predefined Index Values

Some collections have predefined index values you can use to return single objects. Each predefined index value is represented by a constant. For example, you specify an **XlBordersIndex** constant with the **Item** property of the **Borders** collection to return a single border.

The following example sets the bottom border of cells A1:G1 on Sheet1 to a double line.

```
Worksheets("Sheet1").Range("A1:A1")._
    Borders.Item(xlEdgeBottom).LineStyle = xlDouble
```



# Formatting Codes for Headers and Footers

The following special formatting codes can be included as a part of the header and footer properties ([LeftHeader](#), [CenterHeader](#), [RightHeader](#), [LeftFooter](#), [CenterFooter](#), [RightFooter](#) ).

Format code	Description
&L	Left aligns the characters that follow.
&C	Centers the characters that follow.
&R	Right aligns the characters that follow.
&E	Turns double-underline printing on or off.
&X	Turns superscript printing on or off.
&Y	Turns subscript printing on or off.
&B	Turns bold printing on or off.
&I	Turns italic printing on or off.
&U	Turns underline printing on or off.
&S	Turns strikethrough printing on or off.
&D	Prints the current date.
&T	Prints the current time.
&F	Prints the name of the document.
&A	Prints the name of the workbook tab.
&P	Prints the page number.
&P+number	Prints the page number plus the specified number.
&P-number	Prints the page number minus the specified number.
&&	Prints a single ampersand.
& "fontname"	Prints the characters that follow in the specified font. Be sure to include the double quotation marks.
&nn	Prints the characters that follow in the specified font size. Use a two-digit number to specify a size in points.

&N

Prints the total number of pages in the document.

---

---

---

# Using Events with Embedded Charts

Events are enabled for chart sheets by default. Before you can use events with a **Chart** object that represents an embedded chart, you must create a new class module and declare an object of type **Chart** with events. For example, assume that a new class module is created and named EventClassModule. The new class module contains the following code.

```
Public WithEvents myChartClass As Chart
```

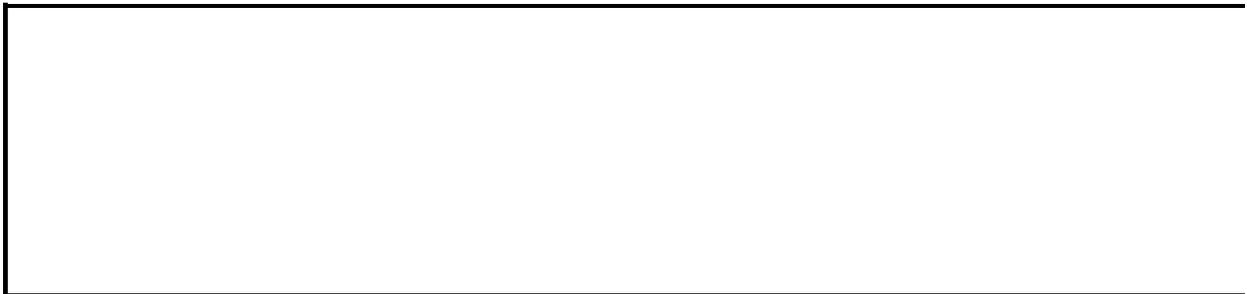
After the new object has been declared with events, it appears in the **Object** drop-down list box in the class module, and you can write event procedures for this object. (When you select the new object in the **Object** box, the valid events for that object are listed in the **Procedure** drop-down list box.)

Before your procedures will run, however, you must connect the declared object in the class module with the embedded chart. You can do this by using the following code from any module.

```
Dim myClassModule As New EventClassModule

Sub InitializeChart()
    Set myClassModule.myChartClass = _
        Worksheets(1).ChartObjects(1).Chart
End Sub
```

After you run the InitializeChart procedure, the myChartClass object in the class module points to embedded chart one on worksheet one, and the event procedures in the class module will run when the events occur.



# Using Events with the Application Object

Before you can use events with the **Application** object, you must create a new class module and declare an object of type **Application** with events. For example, assume that a new class module is created and called EventClassModule. The new class module contains the following code.

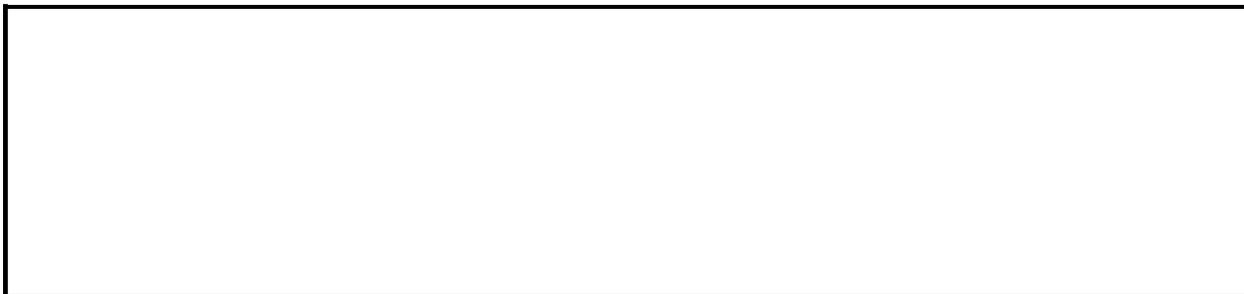
```
Public WithEvents App As Application
```

After the new object has been declared with events, it appears in the **Object** drop-down list box in the class module, and you can write event procedures for the new object. (When you select the new object in the **Object** box, the valid events for that object are listed in the **Procedure** drop-down list box.)

Before the procedures will run, however, you must connect the declared object in the class module with the **Application** object. You can do this with the following code from any module.

```
Dim X As New EventClassModule  
  
Sub InitializeApp()  
    Set X.App = Application  
End Sub
```

After you run the InitializeApp procedure, the App object in the class module points to the Microsoft Excel **Application** object, and the event procedures in the class module will run when the events occur.



# Using ActiveX Controls on a Document

Just as you can add ActiveX controls to [custom dialog boxes](#), you can add controls directly to a document when you want to provide a sophisticated way for the user to interact directly with your macro without the distraction of dialog boxes. Use the following procedure to add ActiveX controls to your document. For more specific information about using ActiveX controls in Microsoft Excel, see [Using ActiveX Controls on Sheets](#).

1. [Add controls to the document](#)

Display the **Control Toolbox**, click the control you want to add, and then click the document.

2. [Set control properties](#)

Right-click a control in design mode and click **Properties** to display the Properties window.

3. [Initialize the controls](#)

You can initialize controls in a procedure.

4. [Write event procedures](#)

All controls have a predefined set of events. For example, a command button has a Click event that occurs when the user clicks the command button. You can write event procedures that run when the events occur.

5. [Use control values while code is running](#)

Some properties can be set at run time.



# Creating a Custom Dialog Box

Use the following procedure to create a custom dialog box:

1. [Create a UserForm](#)

On the **Insert** menu in the Visual Basic Editor, click **UserForm**.

2. [Add controls to the UserForm](#)

Find the control you want to add in the **Toolbox** and drag the control onto the form.

3. [Set control properties](#)

Right-click a control in design mode and click **Properties** to display the Properties window.

4. [Initialize the controls](#)

You can initialize controls in a procedure before you show a form, or you can add code to the Initialize event of the form.

5. [Write event procedures](#)

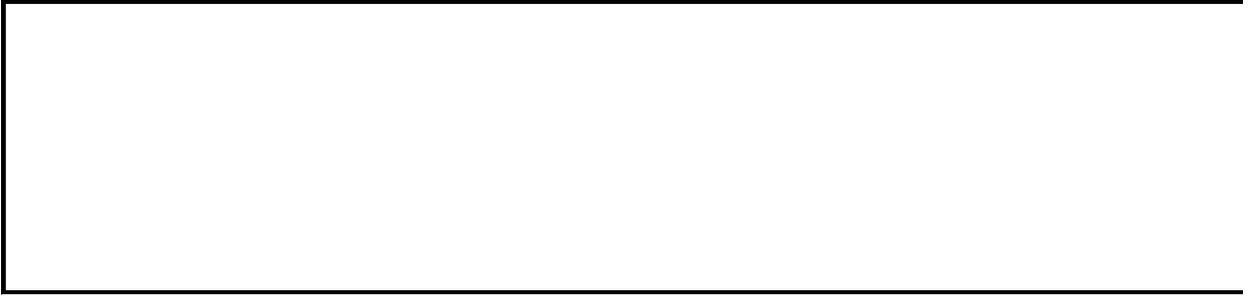
All controls have a predefined set of events. For example, a command button has a Click event that occurs when the user clicks the command button. You can write event procedures that run when the events occur.

6. [Show the dialog box](#)

Use the **Show** method to display a UserForm.

7. [Use control values while code is running](#)

Some properties can be set at run time. Changes made to the dialog box by the user are lost when the dialog box is closed.



# List of Worksheet Functions Available to Visual Basic

A B C D E F G H I J K L M

N O P Q R S T U V W X Y Z

## **A**

[Acos](#)

[Acosh](#)

[And](#)

[Asin](#)

[Asinh](#)

[Atan2](#)

[Atanh](#)

[AveDev](#)

[Average](#)

## **B**

[BetaDist](#)

[BetaInv](#)

[BinomDist](#)

## **C**

[Ceiling](#)

[ChiDist](#)

[ChiInv](#)

[ChiTest](#)

[Choose](#)

[Clean](#)

[Combin](#)

[Confidence](#)

[Correl](#)

[Cosh](#)

[Count](#)

[CountA](#)

[CountBlank](#)

[CountIf](#)

[Covar](#)

[CritBinom](#)

## **D**

[DAverage](#)

[Days360](#)

[Db](#)

[DCount](#)

[DCountA](#)

[Ddb](#)

[Degrees](#)

[DevSq](#)

[DGet](#)

[DMax](#)

[DMin](#)

[Dollar](#)

[DProduct](#)

[DStDev](#)

[DStDevP](#)

[DSum](#)

[DVar](#)

[DVarP](#)

**E**

[Even](#)

[ExponDist](#)

**F**

[Fact](#)

[FDist](#)

[Find](#)

[FindB](#)

[FInv](#)

[Fisher](#)

[FisherInv](#)

[Fixed](#)

[Floor](#)

[Forecast](#)

[Frequency](#)

[FTest](#)

[Fv](#)

**G**

[GammaDist](#)

[GammaInv](#)

[GammaLn](#)

[GeoMean](#)

[Growth](#)

**H**

[HarMean](#)

[HLookup](#)

[HypGeomDist](#)

**I**

[Index](#)

[Intercept](#)

[Ipmt](#)

[Irr](#)

[IsErr](#)

[IsError](#)

[IsLogical](#)

[IsNA](#)

[IsNonText](#)

[IsNumber](#)

[Ispmt](#)

[IsText](#)

**J**

## **K**

[Kurt](#)

## **L**

[Large](#)

[LinEst](#)

[Ln](#)

[Log](#)

[Log10](#)

[LogEst](#)

[LogInv](#)

[LogNormDist](#)

[Lookup](#)

## **M**

[Match](#)

[Max](#)

[MDeterm](#)

[Median](#)

[Min](#)

[MInverse](#)

[MIrr](#)

[MMult](#)

[Mode](#)

**N**

[NegBinomDist](#)

[NormDist](#)

[NormInv](#)

[NormSDist](#)

[NormSInv](#)

[NPer](#)

[Npv](#)

**O**

[Odd](#)

[Or](#)

**P**

[Pearson](#)

[Percentile](#)

[PercentRank](#)

[Permut](#)

[Phonetic](#)

[Pi](#)

[Pmt](#)

[Poisson](#)

[Power](#)

[Ppmt](#)

[Prob](#)

[Product](#)

[Proper](#)

[Pv](#)

**Q**

[Quartile](#)

**R**

[Radians](#)

[Rank](#)

[Rate](#)

[Replace](#)

[ReplaceB](#)

[Rept](#)

[Roman](#)

[Round](#)

[RoundDown](#)

[RoundUp](#)

[RSq](#)

[RTD](#)

**S**

[Search](#)

[SearchB](#)

[Sinh](#)

[Skew](#)

[Sln](#)

[Slope](#)

[Small](#)

[Standardize](#)

[StDev](#)

[StDevP](#)

[StEyx](#)

[Substitute](#)

[Subtotal](#)

[Sum](#)

[SumIf](#)

[SumProduct](#)

[SumSq](#)

[SumX2MY2](#)

[SumX2PY2](#)

[SumXMY2](#)

[Syd](#)

**T**

[Tanh](#)

[TDist](#)

[Text](#)

[TInv](#)

[Transpose](#)

[Trend](#)

[Trim](#)

[TrimMean](#)

[TTest](#)

**U**

[USDollar](#)

**V**

[Var](#)

[VarP](#)

[Vdb](#)

[VLookup](#)

**W**

[Weekday](#)

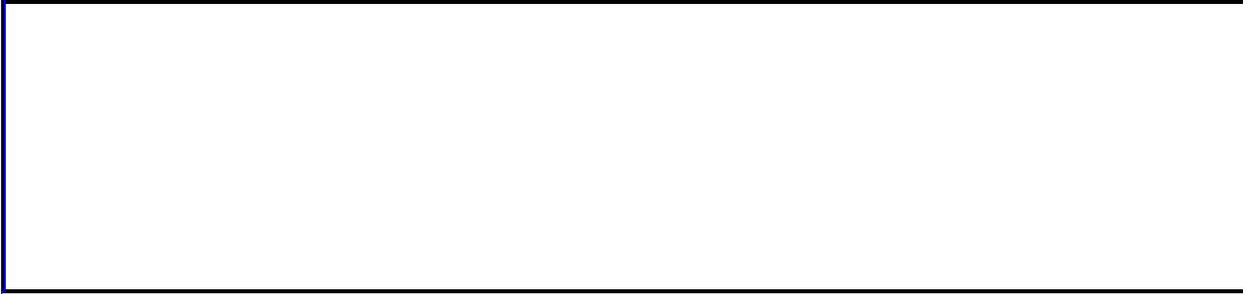
[Weibull](#)

**X**

**Y**

**Z**

[ZTest](#)



# Adding Controls to a Document

To add [controls](#) to a document, display the **Control Toolbox**, click the control you want to add, and then click on the document. Drag an adjustment handle of the control until the control's outline is the size and shape you want.

**Note** Dragging a control (or a number of "grouped" controls) from the form back to the **Control Toolbox** creates a template of that control, which can be reused. This is a useful feature for implementing a standard interface for your applications.



# Setting Control Properties

You can set some [control](#) properties at design time (before any macro is running). In design mode, right-click a control and click **Properties** to display the Properties window. Property names are shown in the left column in the window, property values in the right column. You set a property value by entering the new value to the right of the property name.



# Initializing Control Properties

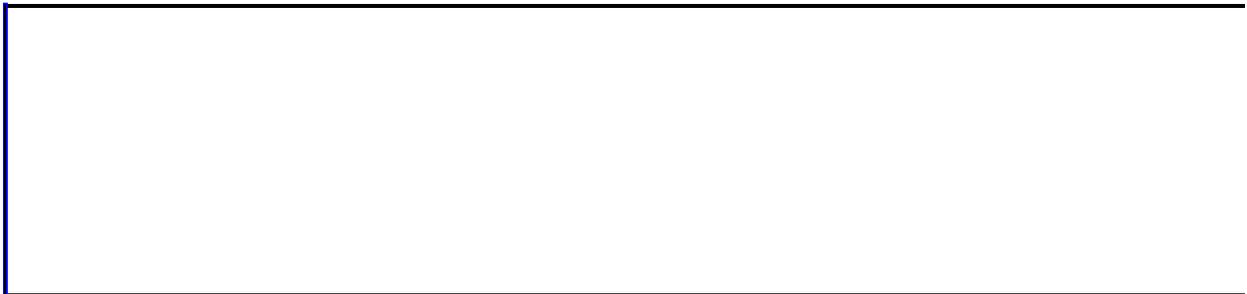
You can initialize [controls](#) at run time by using Visual Basic code in a macro. For example, you could fill a list box, set text values, or set option buttons.

The following example uses the **AddItem** method to add data to a list box. Then it sets the value of a text box and displays the form.

```
Private Sub GetUserName()  
    With UserForm1  
        .lstRegions.AddItem "North"  
        .lstRegions.AddItem "South"  
        .lstRegions.AddItem "East"  
        .lstRegions.AddItem "West"  
        .txtSalesPersonID.Text = "00000"  
        .Show  
        ' ...  
    End With  
End Sub
```

You can also use code in the Initialize event of a form to set initial values for controls on the form. An advantage to setting initial control values in the Initialize event is that the initialization code stays with the form. You can copy the form to another project, and when you run the **Show** method to display the dialog box, the controls will be initialized.

```
Private Sub UserForm_Initialize()  
    UserForm1.lstNames.AddItem "Test One"  
    UserForm1.lstNames.AddItem "Test Two"  
    UserForm1.txtUserName.Text = "Default Name"  
End Sub
```



# Control and Dialog Box Events

After you have added [controls](#) to your dialog box or document, you add event procedures to determine how the controls respond to user actions.

User forms and controls have a predefined set of events. For example, a command button has a Click event that occurs when the user clicks the command button, and UserForms have an Initialize event that runs when the form is loaded.

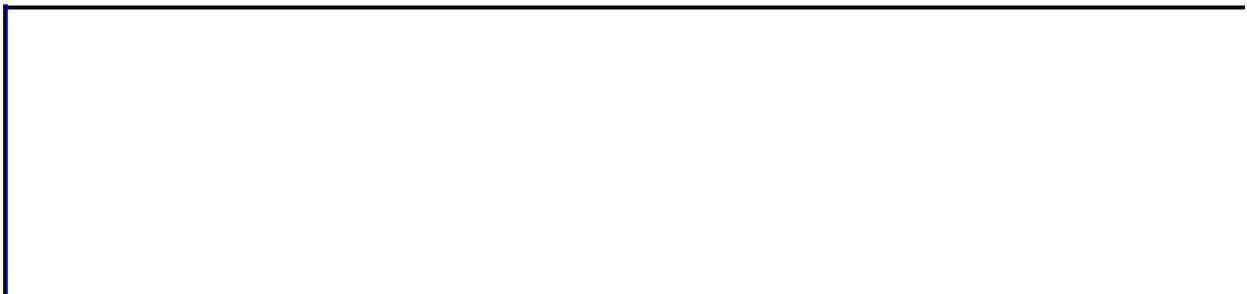
To write a control or form event procedure, open a module by double-clicking the form or control, and select the event from the **Procedure** drop-down list box.

Event procedures include the name of the control. For example, the name of the Click event procedure for a command button named Command1 is Command1\_Click.

If you add code to an event procedure and then change the name of the control, your code remains in procedures with the previous name.

For example, assume you add code to the Click event for Command1 and then rename the control to Command2. When you double-click Command2, you will not see any code in the Click event procedure. You will need to move code from Command1\_Click to Command2\_Click.

To simplify development, it's a good practice to name your controls before writing code.



# Using Control Values While Code Is Running

Some [control](#) properties can be set and returned while Visual Basic code is running. The following example sets the **Text** property of a text box to "Hello."

```
TextBox1.Text = "Hello"
```

The data entered on a form by a user is lost when the form is closed. If you return the values of controls on a form after the form has been unloaded, you get the initial values for the controls rather than the values the user entered.

If you want to save the data entered on a form, you can save the information to module-level variables while the form is still running. The following example displays a form and saves the form data.

```
' Code in module to declare public variables.
Public strRegion As String
Public intSalesPersonID As Integer
Public blnCancelled As Boolean

' Code in form.
Private Sub cmdCancel_Click()
    Module1.blnCancelled = True
    Unload Me
End Sub

Private Sub cmdOK_Click()
    ' Save data.
    intSalesPersonID = txtSalesPersonID.Text
    strRegion = lstRegions.List(lstRegions.ListIndex)
    Module1.blnCancelled = False
    Unload Me
End Sub

Private Sub UserForm_Initialize()
    Module1.blnCancelled = True
End Sub

' Code in module to display form.
Sub LaunchSalesPersonForm()
```

```
frmSalesPeople.Show
If blnCancelled = True Then
    MsgBox "Operation Cancelled!", vbExclamation
Else
    MsgBox "The Salesperson's ID is: " &
        intSalesPersonID & _
        "The Region is: " & strRegion
End If
End Sub
```



# Creating a User Form

To create a custom dialog box, you must create a UserForm. To create a UserForm, click **UserForm** on the **Insert** menu in the Visual Basic Editor.

Use the **Properties** window to change the name, behavior, and appearance of the form. For example, to change the caption on a form, set the **Caption** property.



# Adding Controls to a User Form

To add [controls](#) to a user form, find the control you want to add in the **ToolBox**, drag the control onto the form, and then drag an adjustment handle on the control until the control's outline is the size and shape you want.

**Note** Dragging a control (or a number of "grouped" controls) from the form back to the **Toolbox** creates a template of that control, which can be reused. This is a useful feature for implementing a standard interface for your applications.

When you've added controls to the form, use the commands on the **Format** menu in the Visual Basic Editor to adjust the control alignment and spacing.



# Displaying a Custom Dialog Box

To test your dialog box in the Visual Basic Editor, click **Run Sub/UserForm** on the **Run** menu in the Visual Basic Editor.

To display a dialog box from Visual Basic, use the **Show** method. The following example displays the dialog box named UserForm1.

```
Private Sub GetUserName()  
    UserForm1.Show  
End Sub
```



# ActiveX Controls

For more information on a specific control, select an object from the following list. For information about events, select a control and click Events at the top of the topic.

[CheckBox](#)

[OptionButton](#)

[ComboBox](#)

[ScrollBar](#)

[CommandButton](#)

[SpinButton](#)

[Image](#)

[TextBox](#)

[Label](#)

[ToggleButton](#)

[ListBox](#)

