CodePanes Collection
Contains the active code panes in the VBE object.

**Remarks**

Use the **CodePanes** collection to access the open code panes in a project.

You can use the **Count** property to return the number of active code panes in a collection.
CommandBars Collection
Contains all of the command bars in a project, including command bars that support shortcut menus.

Remarks

Use the CommandBars collection to enable add-ins to add command bars and controls, or to add controls to existing, built-in, command bars.
LinkedWindows Collection
Contains all linked windows in a linked window frame.

**Remarks**

Use the `LinkedWindows` collection to modify the `docked` and `linked` state of windows in the development environment.

**Important** Objects, properties, and methods for controlling linked windows, linked window frames, and docked windows are included on the Macintosh for compatibility with code written in Windows. However, these language elements will generate run-time errors when run on the Macintosh.

The `LinkedWindowFrame` property of the `Window` object returns a `Window` object that has a valid `LinkedWindows` collection.

Linked window frames contain all windows that can be linked or docked. This includes all windows except code windows, `designers`, the `Object Browser` window, and the `Search and Replace` window.

If all the panes from one linked window frame are moved to another window, the linked window frame with no panes is destroyed. However, if all the panes are removed from the main window, it isn't destroyed.
Use the **Visible** property to check or set the visibility of a window.

You can use the **Add** method to add a window to the **collection** of currently linked windows. A window that is a pane in one linked window frame can be added to another linked window frame. Use the **Remove** method to remove a window from the collection of currently linked windows; this results in the window being unlinked or undocked.

The **LinkedWindows** collection is used to dock and undock windows from the main window frame.
Properties Collection (VBA Add-In Object Model)

**Properties Collection**  
- **Property Object**

Represents the *properties* of an object.

**Remarks**

Use the **Properties** collection to access the properties displayed in the **Properties window**. For every property listed in the **Properties** window, there is an object in the **Properties** collection.
References Collection

Represents the set of references in the project.

Remarks

Use the References collection to add or remove references. The References collection is the same as the set of references selected in the References dialog box.
Represents the components contained in a project.

Remarks

Use the **VBComponents** collection to access, add, or remove components in a project. A component can be a **form**, **module**, or **class**. The **VBComponents** collection is a standard **collection** that can be used in a **For Each** block.

You can use the **Parent** property to return the project the **VBComponents** collection is in.

In Visual Basic for Applications, you can use **Import** method to add a component to a project from a file.
VBProjects Collection

Represented all the projects that are open in the development environment.

Remarks

Use the VBProjects collection to access specific projects in an instance of the development environment. VBProjects is a standard collection that can be used in a For Each block.
Windows Collection
Contains all open or permanent windows.

**Remarks**

Use the **Windows** collection to access **Window** objects.

The **Windows** collection has a fixed set of windows that are always available in the **collection**, such as the **Project window**, the **Properties window**, and a set of windows that represent all open code windows and **designer** windows. Opening a code or designer window adds a new member to the **Windows** collection. Closing a code or designer window removes a member from the **Windows** collection. Closing a permanent **development environment** window doesn't remove the corresponding object from this collection, but results in the window not being visible.
Click Event (VBA Add-In Object Model)
Occurs when the **OnAction** property of a corresponding command bar control is set.

**Syntax**

```vba
Sub object_Click (ByVal ctrl As Object, ByRef handled As Boolean, ByRef canceldefault As Boolean)
```

The Click event syntax has these **named arguments**:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ctrl</strong></td>
<td>Required; <strong>Object</strong>. Specifies the object that is the source of the Click event.</td>
</tr>
<tr>
<td><strong>handled</strong></td>
<td>Required; <strong>Boolean</strong>. If <strong>True</strong>, other add-ins should handle the event. If <strong>False</strong>, the action of the command bar item has not been handled.</td>
</tr>
<tr>
<td><strong>canceldefault</strong></td>
<td>Required; <strong>Boolean</strong>. If <strong>True</strong>, default behavior is performed unless canceled by a downstream add-in. If <strong>False</strong>, default behavior is not performed unless restored by a downstream add-in.</td>
</tr>
</tbody>
</table>

**Remarks**

The Click event is specific to the **CommandBarEvents** object. Use a **variable** declared using the ** WithEvents** keyword to receive the Click event for a **CommandBar** control. This variable should be set to the return value of the **CommandBarEvents** property of the **Events** object. The **CommandBarEvents** property takes the **CommandBar** control as an **argument**. When the **CommandBar** control is clicked (for the variable you declared using the ** WithEvents** keyword), the code is executed.
Occurs after a reference is added.

**Syntax**

```vba
Sub object_ItemAdded(ByVal item As Reference)
```

The required *item* argument specifies the item that was added.

**Remarks**

The ItemAdded event occurs when a Reference is added to the References collection.
Occurs after a reference is removed from a project.

Syntax

Sub object_ItemRemoved(ByVal item As Reference)

The required item argument specifies the Reference that was removed.
Add Method (VBA Add-In Object Model)
Adds an object to a collection.

Syntax

```
object.Add(component)
```

The Add syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Required. An object expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td><code>component</code></td>
<td>Required. For the LinkedWindows collection, an object. For the VBComponents collection, an enumerated constant representing a class module, a form, or a standard module. For the VBProjects collection, an enumerated constant representing a project type.</td>
</tr>
</tbody>
</table>

You can use one of the following constants for the `component` argument:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vbext_ct_ClassModule</code></td>
<td>Adds a class module to the collection.</td>
</tr>
<tr>
<td><code>vbext_ct_MSForm</code></td>
<td>Adds a form to the collection.</td>
</tr>
<tr>
<td><code>vbext_ct_StdModule</code></td>
<td>Adds a standard module to the collection.</td>
</tr>
<tr>
<td><code>vbext_pt_Standalone</code></td>
<td>Adds a standalone project to the collection.</td>
</tr>
</tbody>
</table>

Remarks

For the LinkedWindows collection, the Add method adds a window to the
collection of currently linked windows.

**Note** You can add a window that is a pane in one linked window frame to another linked window frame; the window is simply moved from one pane to the other. If the linked window frame that the window was moved from no longer contains any panes, it's destroyed.

**Important** Objects, properties, and methods for controlling linked windows, linked window frames, and docked windows are included on the Macintosh for compatibility with code written in Windows. However, these language elements generate run-time errors when run on the Macintosh.

For the **VBComponents** collection, the **Add** method creates a new standard component and adds it to the **project**.

For the **VBComponents** collection, the **Add** method returns a **VBComponent** object. For the **LinkedWindows** collection, the **Add** method returns **Nothing**.

For the **VBProjects** collection, the **Add** method returns a **VBProject** object and adds a project to the **VBProjects** collection.
AddFromFile Method

For the **References** collection, adds a reference to a **project** from a file. For the **CodeModule** object, adds the contents of a file to a **module**.
Syntax

`object.AddFromFile(filename)`

The `AddFromFile` syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Required. An <code>object expression</code> that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td><code>filename</code></td>
<td>Required. A <code>string expression</code> specifying the name of the file you want to add to the project or module. If the file name isn't found and a path name isn't specified, the directories searched by the <code>Windows OpenFile</code> function are searched.</td>
</tr>
</tbody>
</table>

Remarks

For the `CodeModule` object, the `AddFromFile` method inserts the contents of the file starting on the line preceding the first `procedure` in the `code module`. If the module doesn't contain procedures, `AddFromFile` places the contents of the file at the end of the module.
AddFromGuid Method

Adds a reference to the References collection using the globally unique identifier (GUID) of the reference.
Syntax

*object*.AddFromGuid(*guid*, *major*, *minor*) *As Reference*

The *AddFromGuid* syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>object</em></td>
<td>Required. An <em>object expression</em> that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td><em>guid</em></td>
<td>Required. A <em>string expression</em> representing the GUID of the reference.</td>
</tr>
<tr>
<td><em>major</em></td>
<td>Required. A <em>Long</em> specifying the major version number of the reference.</td>
</tr>
<tr>
<td><em>minor</em></td>
<td>Required. A <em>Long</em> specifying the minor version number of the reference.</td>
</tr>
</tbody>
</table>

Remarks

The *AddFromGuid* method searches the registry to find the reference you want to add. The GUID can be a *type library*, control, class identifier, and so on.
AddFromString Method

Add text to a module.

Syntax

```
object.AddFromString
```

The `object` placeholder is an object expression that evaluates to an object in the Applies To list.
Remarks

The **AddFromString** method inserts the text starting on the line preceding the first **procedure** in the module. If the module doesn't contain procedures, **AddFromString** places the text at the end of the module.
Closes and destroys a window.

**Syntax**

`object.Close`

The *object* placeholder is an *object expression* that evaluates to an object in the Applies To list.
Remarks

The following types of windows respond to the Close method in different ways:

For a window that is a code pane, Close destroys the code pane.

For a window that is a designer, Close destroys the contained designer.

For windows that are always available on the View menu, Close hides the window.
CreateEventProc Method

Creates an event procedure.

Syntax
**object.CreateEventProc(eventname, objectname) As Long**

The **CreateEventProc** syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Required. An <a href="#">object expression</a> that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>eventname</td>
<td>Required. A <a href="#">string expression</a> specifying the name of the event you want to add to the module.</td>
</tr>
<tr>
<td>objectname</td>
<td>Required. A string expression specifying the name of the object that is the source of the event.</td>
</tr>
</tbody>
</table>

**Remarks**

Use the **CreateEventProc** method to create an event procedure. For example, to create an event procedure for the Click event of a **Command Button** control named Command1 you would use the following code, where cm represents a object of type **CodeModule**:

```
TextLocation = CM.CreateEventProc("Click", "Command1")
```

The **CreateEventProc** method returns the line at which the body of the event procedure starts. **CreateEventProc** fails if the arguments refer to a nonexistent event.
DeleteLines Method

Deletes a single line or a specified range of lines.

**Syntax**

`object.DeleteLines (startline [, count])`

The **DeleteLines** syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Required. An <code>object expression</code> that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td><code>startline</code></td>
<td>Required. A <code>Long</code> specifying the first line you want to delete.</td>
</tr>
<tr>
<td><code>count</code></td>
<td>Optional. A <code>Long</code> specifying the number of lines you want to delete.</td>
</tr>
</tbody>
</table>
Remarks

If you don't specify how many lines you want to delete, **DeleteLines** deletes one line.
Exports a component as a separate file or files.

**Syntax**

`object.Export(filename)`

The `Export` syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Required. An <em>object expression</em> that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td><code>filename</code></td>
<td>Required. A <em>String</em> specifying the name of the file that you want to export the component to.</td>
</tr>
</tbody>
</table>

**Remarks**

When you use the `Export` method to save a component as a separate file or files,
use a file name that doesn't already exist; otherwise, an error occurs.
Find Method (VBA Add-In Object Model)

Searches the active module for a specified string.
Syntax

```plaintext
object.Find(target, startline, startcol, endline, endcol [, wholeword]
[, matchcase] [, patternsearch]) As Boolean
```

The `Find` syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Required. An <strong>object expression</strong> that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td><code>target</code></td>
<td>Required. A <strong>String</strong> containing the text or pattern you want to find.</td>
</tr>
<tr>
<td><code>startline</code></td>
<td>Required. A <strong>Long</strong> specifying the line at which you want to start the search; will be set to the line of the match if one is found. The first line is number 1.</td>
</tr>
<tr>
<td><code>startcol</code></td>
<td>Required. A <strong>Long</strong> specifying the column at which you want to start the search; will be set to the column containing the match if one is found. The first column is 1.</td>
</tr>
<tr>
<td><code>endline</code></td>
<td>Required. A <strong>Long</strong> specifying the last line of the match if one is found. The last line may be specified as –1.</td>
</tr>
<tr>
<td><code>endcol</code></td>
<td>Required. A <strong>Long</strong> specifying the last line of the match if one is found. The last column may be designated as –1.</td>
</tr>
<tr>
<td><code>wholeword</code></td>
<td>Optional. A <strong>Boolean</strong> value specifying whether to only match whole words. If <strong>True</strong>, only matches whole words. <strong>False</strong> is the default.</td>
</tr>
<tr>
<td><code>matchcase</code></td>
<td>Optional. A <strong>Boolean</strong> value specifying whether to match case. If <strong>True</strong>, the search is case sensitive. <strong>False</strong> is the default.</td>
</tr>
<tr>
<td><code>patternsearch</code></td>
<td>Optional. A <strong>Boolean</strong> value specifying whether or not the target string is a regular expression pattern. If <strong>True</strong>, the target string is a regular expression pattern. <strong>False</strong> is the default.</td>
</tr>
</tbody>
</table>

**Remarks**

`Find` returns **True** if a match is found and **False** if a match isn't found.

The `matchcase` and `patternsearch` arguments are mutually exclusive; if both
arguments are passed as **True**, an error occurs.

The content of the **Find** dialog box isn't affected by the **Find** method.

The specified range of lines and columns is inclusive, so a search can find the pattern on the specified last line if *endcol* is supplied as either –1 or the length of the line.
GetSelection Method

Returns the selection in a code pane.

Syntax
object.GetSelection(startline, startcol, endline, endcol)

The GetSelection syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Required. An object expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>startline</td>
<td>Required. A Long that returns a value specifying the first line of the selection in the code pane.</td>
</tr>
<tr>
<td>startcol</td>
<td>Required. A Long that returns a value specifying the first column of the selection in the code pane.</td>
</tr>
<tr>
<td>endline</td>
<td>Required. A Long that returns a value specifying the last line of the selection in the code pane.</td>
</tr>
<tr>
<td>endcol</td>
<td>Required. A Long that returns a value specifying the last column of the selection in the code pane.</td>
</tr>
</tbody>
</table>

Remarks

When you use the GetSelection method, information is returned in output arguments. As a result, you must pass in variables because the variables will be modified to contain the information when returned.
 Adds a component to a project from a file; returns the newly added component.
Syntax

`object.Import(filename) As VBComponent`

The `Import` syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Required. An <code>object expression</code> that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td><code>filename</code></td>
<td>Required. A <code>String</code> specifying path and file name of the component that you want to import the component from.</td>
</tr>
</tbody>
</table>

Remarks

You can use the `Import` method to add a component, `form`, `module`, `class`, and so on, to your project.
InsertLines Method

Inserts a line or lines of code at a specified location in a block of code.

Syntax

```
object.InsertLines(line, code)
```

The **InsertLines** syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
</table>
**object**
Required. An **object expression** that evaluates to an object in the Applies To list.

**line**
Required. A **Long** specifying the location at which you want to insert the code.

**code**
Required. A **String** containing the code you want to insert.

**Remarks**

If the text you insert using the **InsertLines** method is carriage return–linefeed delimited, it will be inserted as consecutive lines.
Item Method (VBA Add-In Object Model)

Returns the indexed member of a collection.

Syntax
**object.Item(index)**

The **Item** method syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Required. An <strong>object expression</strong> that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>index</td>
<td>Required. An expression that specifies the position of a member of the collection. If a <strong>numeric expression</strong>, <code>index</code> must be a number from 1 to the value of the collection's <strong>Count</strong> property. If a <strong>string expression</strong>, <code>index</code> must correspond to the <strong>key argument</strong> specified when the member was added to the collection.</td>
</tr>
</tbody>
</table>

The following table lists the collections and their corresponding **key arguments** for use with the **Item** method. The string you pass to the **Item** method must match the collection's **key argument**.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Key argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td><strong>Caption</strong> property setting</td>
</tr>
<tr>
<td>LinkedWindows</td>
<td><strong>Caption</strong> property setting</td>
</tr>
<tr>
<td>CodePanes</td>
<td>No unique string is associated with this collection.</td>
</tr>
<tr>
<td>VBProjects</td>
<td><strong>Name</strong> property setting</td>
</tr>
<tr>
<td>VBComponents</td>
<td><strong>Name</strong> property setting</td>
</tr>
<tr>
<td>References</td>
<td><strong>Name</strong> property setting</td>
</tr>
<tr>
<td>Properties</td>
<td><strong>Name</strong> property setting</td>
</tr>
</tbody>
</table>

**Remarks**

The `index` argument can be a numeric value or a string containing the title of the object.

**Important** Objects, properties, and methods for controlling linked windows, linked window frames, and docked windows are included on the Macintosh for
compatibility with code written in Windows. However, these language elements will generate run-time errors when run on the Macintosh.
ProcBodyLine Property
Returns the first line of a **procedure**.

**Syntax**

```
object.ProcBodyLine(procname, prockind) As Long
```

The **ProcBodyLine** syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Required. An <strong>object expression</strong> that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>procname</td>
<td>Required. A <strong>String</strong> containing the name of the procedure.</td>
</tr>
<tr>
<td>prockind</td>
<td>Required. Specifies the kind of procedure to locate. Because <strong>property procedures</strong> can have multiple representations in the <strong>module</strong>, you must specify the kind of procedure you want to locate. All procedures other than property procedures (that is, <strong>Sub</strong> and <strong>Function</strong> procedures) use <strong>vbext_pk_Proc</strong>.</td>
</tr>
</tbody>
</table>

You can use one of the following **constants** for the **prockind** argument:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbext_pk_Get</td>
<td>Specifies a procedure that returns the value of a property.</td>
</tr>
<tr>
<td>vbext_pk_Let</td>
<td>Specifies a procedure that assigns a value to a property.</td>
</tr>
<tr>
<td>vbext_pk_Set</td>
<td>Specifies a procedure that sets a reference to an object.</td>
</tr>
<tr>
<td>vbext_pk_Proc</td>
<td>Specifies all procedures other than property procedures.</td>
</tr>
</tbody>
</table>

**Remarks**
The first line of a procedure is the line on which the **Sub, Function, or Property** statement appears.
ProcCountLines Property
Returns the number of lines in the specified procedure.

**Syntax**

`object.ProcCountLines(procname, prockind) As Long`

The **ProcCountLines** syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Required. An <em>object expression</em> that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td>procname</td>
<td>Required. A <em>String</em> containing the name of the procedure.</td>
</tr>
<tr>
<td>prockind</td>
<td>Required. Specifies the kind of procedure to locate. Because <em>property procedures</em> can have multiple representations in the <em>module</em>, you must specify the kind of procedure you want to locate. All procedures other than property procedures (that is, <em>Sub</em> and <em>Function</em> procedures) use <code>vbext_pk_Proc</code>.</td>
</tr>
</tbody>
</table>

You can use one of the following *constants* for the *prockind* argument:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbext_pk_Get</td>
<td>Specifies a procedure that returns the value of a property.</td>
</tr>
<tr>
<td>vbext_pk_Let</td>
<td>Specifies a procedure that assigns a value to a property.</td>
</tr>
<tr>
<td>vbext_pk_Set</td>
<td>Specifies a procedure that sets a reference to an object.</td>
</tr>
<tr>
<td>vbext_pk_Proc</td>
<td>Specifies all procedures other than property procedures.</td>
</tr>
</tbody>
</table>

**Remarks**
The **ProcCountLines** property returns the count of all blank or comment lines preceding the procedure declaration and, if the procedure is the last procedure in a code module, any blank lines following the procedure.
ProcOfLine Property
Returns the name of the procedure that the specified line is in.

**Syntax**

```
object.ProcOfLine(line, prockind) As String
```

The `ProcOfLine` syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Required. An object expression that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td><code>line</code></td>
<td>Required. A <code>Long</code> specifying the line to check.</td>
</tr>
<tr>
<td><code>prockind</code></td>
<td>Required. Specifies the kind of procedure to locate. Because property procedures can have multiple representations in the module, you must specify the kind of procedure you want to locate. All procedures other than property procedures (that is, <code>Sub</code> and <code>Function</code> procedures) use <code>vbext_pk_Proc</code>.</td>
</tr>
</tbody>
</table>

You can use one of the following constants for the `prockind` argument:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vbext_pk_Get</code></td>
<td>Specifies a procedure that returns the value of a property.</td>
</tr>
<tr>
<td><code>vbext_pk_Let</code></td>
<td>Specifies a procedure that assigns a value to a property.</td>
</tr>
<tr>
<td><code>vbext_pk_Set</code></td>
<td>Specifies a procedure that sets a reference to an object.</td>
</tr>
<tr>
<td><code>vbext_pk_Proc</code></td>
<td>Specifies all procedures other than property procedures.</td>
</tr>
</tbody>
</table>

**Remarks**
A line is within a procedure if it's a blank line or comment line preceding the procedure declaration and, if the procedure is the last procedure in a code module, a blank line or lines following the procedure.
ProcStartLine Property
Returns the line at which the specified procedure begins.

**Syntax**

`object.ProcStartLine(procname, prockind) As Long`

The *ProcStartLine* syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Required. An <em>object expression</em> that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td><code>procname</code></td>
<td>Required. A <em>String</em> containing the name of the procedure.</td>
</tr>
<tr>
<td><code>prockind</code></td>
<td>Required. Specifies the kind of procedure to locate. Because <em>property procedures</em> can have multiple representations in the module, you must specify the kind of procedure you want to locate. All procedures other than property procedures (that is, <em>Sub</em> and <em>Function</em> procedures) use <em>vbext_pk_Proc</em>.</td>
</tr>
</tbody>
</table>

You can use one of the following *constants* for the *prockind* argument:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>vbext_pk_Get</em></td>
<td>Specifies a <em>procedure</em> that returns the value of a property.</td>
</tr>
<tr>
<td><em>vbext_pk_Let</em></td>
<td>Specifies a procedure that assigns a value to a property.</td>
</tr>
<tr>
<td><em>vbext_pk_Set</em></td>
<td>Specifies a procedure that sets a reference to an object.</td>
</tr>
<tr>
<td><em>vbext_pk_Proc</em></td>
<td>Specifies all procedures other than property procedures.</td>
</tr>
</tbody>
</table>

**Remarks**
A procedure starts at the first line below the **End Sub** statement of the preceding procedure. If the procedure is the first procedure, it starts at the end of the general Declarations section.
Remove Method (VBA Add-In Object Model)
Removes an item from a `collection`.

**Syntax**

`object.Remove(component)`

The `Remove` syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Required. An <code>object expression</code> that evaluates to an object in the Applies To list.</td>
</tr>
<tr>
<td><code>component</code></td>
<td>Required. For the <code>LinkedWindows</code> collection, an object. For the <code>References</code> collection, a reference to a <code>type library</code> or a <code>project</code>. For the <code>VBComponents</code> collection, an enumerated <code>constant</code> representing a <code>class module</code>, a form, or a <code>standard module</code>. For the <code>VBProjects</code> collection, a standalone project.</td>
</tr>
</tbody>
</table>

**Remarks**

When used on the `LinkedWindows` collection, the `Remove` method removes a window from the collection of currently linked windows. The removed window becomes a floating window that has its own linked window frame. The `Remove` method can only be used on a standalone project. It generates a run-time error if you try to use it on a host project.
**Important** Objects, properties, and methods for controlling linked windows, linked window frames, and docked windows are included on the Macintosh for compatibility with code written in Windows. However, these language elements generate run-time errors when run on the Macintosh.
ReplaceLine Method

Replaces an existing line of code with a specified line of code.

Syntax

```
object.ReplaceLine(line, code)
```

The ReplaceLine syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td></td>
</tr>
<tr>
<td>ReplaceLine</td>
<td>line, code</td>
</tr>
<tr>
<td>code</td>
<td></td>
</tr>
</tbody>
</table>
object  Required. An object expression that evaluates to an object in the Applies To list.

line   Required. A Long specifying the location of the line you want to replace.

code   Required. A String containing the code you want to insert.
SetFocus Method

Moves the focus to the specified window.

Syntax

\texttt{object.SetFocus}

The \textit{object} placeholder is an \texttt{object expression} that evaluates to an object in the Applies To list.

Remarks

Use the \texttt{SetFocus} method on windows that are already visible.
Sets the selection in the code pane.

**Syntax**

`object.SetSelection(startline, startcol, endline, endcol)`

The **SetSelection** syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
</table>

object

Required. An object expression that evaluates to an object in the Applies To list.

startline

Required. A Long specifying the first line of the selection.

startcol

Required. A Long specifying the first column of the selection.

endline

Required. A Long specifying the last line of the selection.

endcol

Required. A Long specifying the last column of the selection.
CodeModule Object
Represents the code behind a component, such as a form, class, or document.

Remarks

You use the **CodeModule** object to modify (add, delete, or edit) the code associated with a component.

Each component is associated with one **CodeModule** object. However, a **CodeModule** object can be associated with multiple code panes.

The methods associated with the **CodeModule** object enable you to manipulate and return information about the code text on a line-by-line basis. For example, you can use the **AddFromString** method to add text to the module. **AddFromString** places the text just above the first procedure in the module or places the text at the end of the module if there are no procedures.

Use the **Parent** property to return the **VBComponent** object associated with a code module.
Represents a code pane.

**Remarks**

Use the CodePane object to manipulate the position of visible text or the text selection displayed in the code pane.
You can use the `Show` method to make the code pane you specify visible. Use the `SetSelection` method to set the selection in a code pane and the `GetSelection` method to return the location of the selection in a code pane.
Returned by the **CommandBarEvents** property. The **CommandBarEvents** object triggers an event when a control on the command bar is clicked.

**Remarks**

The **CommandBarEvents** object is returned by the **CommandBarEvents**
property of the Events object. The object that is returned has one event in its interface, the Click event. You can handle this event using the WithEvents object declaration.
Events Object

Supplies properties that enable add-ins to connect to all events in Visual Basic for Applications.
Remarks

The **Events** object provides properties that return *event source objects*. Use the properties to return event source objects that notify you of changes in the Visual Basic for Applications environment.

The properties of the **Events** object return objects of the same type as the property name. For example, the **CommandBarEvents** property returns the **CommandBarEvents** object.
Property Object
Represents the properties of an object that are visible in the Properties window for any given component.

**Remarks**

Use **Value** property of the **Property** object to return or set the value of a property of a component.

At a minimum, all components have a **Name** property. Use the **Value** property of the **Property** object to return or set the value of a property. The **Value** property returns a **Variant** of the appropriate type. If the value returned is an object, the **Value** property returns the **Properties** collection that contains **Property** objects representing the individual properties of the object. You can access each of the **Property** objects by using the **Item** method on the returned **Properties** collection.

If the value returned by the **Property** object is an object, you can use the **Object** property to set the **Property** object to a new object.
Reference Object

Represents a reference to a type library or a project.

Remarks

Use the Reference object to verify whether a reference is still valid.

The IsBroken property returns True if the reference no longer points to a valid
reference. The **BuiltIn** property returns **True** if the reference is a default reference that can't be moved or removed. Use the **Name** property to determine if the reference you want to add or remove is the correct one.
Returned by the **ReferencesEvents** property.

**Remarks**

The **ReferencesEvents** object is the source of events that occur when a reference is added to or removed from a [project](#). The ItemAdded event is triggered after a reference is added to a project. The ItemRemoved event is triggered after a reference is removed from a project.
VBComponent Object
Represents a component, such as a class module or standard module, contained in a project.

Remarks

Use the VBComponent object to access the code module associated with a component or to change a component's property settings.

You can use the Type property to find out what type of component the VBComponent object refers to. Use the Collection property to find out what collection the component is in.
VBE Object
The root object that contains all other objects and collections represented in Visual Basic for Applications.

**Remarks**

You can use the following collections to access the objects contained in the VBE object:

Use the **VBProjects** collection to access the collection of projects.

Use the **AddIns** collection to access the collection of add-ins.

Use the **Windows** collection to access the collection of windows.

Use the **CodePanes** collection to access the collection of code panes.

Use the **CommandBars** collection to access the collection of command bars.

Use the **Events** object to access properties that enable add-ins to connect to all events in Visual Basic for Applications. The properties of the **Events** object
return objects of the same type as the property name. For example, the CommandBarEvents property returns the CommandBarEvents object.

You can use the SelectedVBComponent property to return the active component. The active component is the component that is being tracked in the Project window. If the selected item in the Project window isn't a component, SelectedVBComponent returns Nothing.

**Note** All objects in this object model have a VBE property that points to the VBE object.
VBProject Object

Represents a project.

Remarks

Use the VBProject object to set properties for the project, to access the
VBComponents collection, and to access the References collection.
Window Object
Represents a window in the development environment.

Remarks

Use the **Window** object to show, hide, or position windows.

**Important** Objects, properties, and methods for controlling linked windows, linked window frames, and docked windows are included on the Macintosh for compatibility with code written in Windows. However, these language elements will generate run-time errors when run on the Macintosh.

You can use the **Close** method to close a window in the **Windows** collection. The **Close** method affects different types of windows as follows:

<table>
<thead>
<tr>
<th>Window</th>
<th>Result of using Close method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code window</td>
<td>Removes the window from the <strong>Windows</strong> collection.</td>
</tr>
<tr>
<td><strong>Designer</strong></td>
<td>Removes the window from the <strong>Windows</strong> collection.</td>
</tr>
<tr>
<td><strong>Window</strong> objects of type <strong>linked window frame</strong></td>
<td>Windows become unlinked separate windows.</td>
</tr>
</tbody>
</table>

**Note** Using the **Close** method with code windows and designers actually closes
the window. Setting the **Visible** property to **False** hides the window but doesn't close the window. Using the **Close** method with development environment windows, such as the **Project window** or **Properties window**, is the same as setting the **Visible** property to **False**.

You can use the **SetFocus** method to move the **focus** to a window.

You can use the **Visible** property to return or set the visibility of a window.

To find out what type of window you are working with, you can use the **Type** property. If you have more than one window of a type, for example, multiple designers, you can use the **Caption** property to determine the window you're working with. You can also find the window you want to work with using the **DesignerWindow** property of the **VBComponent** object or the **Window** property of the **CodePane** object.
Returns the active or last active CodePane object or sets the active CodePane object. Read/write.

Remarks

You can set the ActiveCodePane property to any valid CodePane object, as shown in the following example:

```
Set MyApp.VBE.ActiveCodePane = MyApp.VBE.CodePanes(1)
```

The preceding example sets the first code pane in a collection of code panes to be the active code pane. You can also activate a code pane using the Set method.
ActiveVBProject Property

Returns the active project in the Project window. Read-only.

Remarks

The **ActiveVBProject** property returns the project that is selected in the Project window or the project in which the components are selected. In the latter case, the project itself isn't necessarily selected. Whether or not the project is explicitly selected, there is always an active project.
ActiveWindow Property

Returns the active window in the development environment. Read-only.

Remarks

When more than one window is open in the development environment, the ActiveWindow property setting is the window with the focus. If the main window has the focus, ActiveWindow returns Nothing.
Returns a **Boolean** value indicating whether or not the reference is a default reference that can't be removed. Read-only.

**Return Values**

The **BuiltIn** property returns these values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The reference is a default reference that can't be removed.</td>
</tr>
<tr>
<td>False</td>
<td>The reference isn't a default reference; it can be removed.</td>
</tr>
</tbody>
</table>
Returns a String containing the title of the active window. Read-only.

**Remarks**

The title of the active window is the text displayed in the window's title bar.
Returns an object representing the code behind the component. Read-only.

**Remarks**

The **CodeModule** property returns **Nothing** if the component doesn't have a code module associated with it.

**Note**  The **CodePane** object represents a visible code window. A given component can have several **CodePane** objects. The **CodeModule** object represents the code within a component. A component can only have one **CodeModule** object.
Returns a **CodePane** object. Read-only.

**Remarks**

If a [code pane](#) exists, it becomes the active code pane, and the window that contains it becomes the active window. If a code pane doesn't exist for the [module](#), the **CodePane** property creates one.
CodePanes Property

Returns the collection of active CodePane objects. Read-only.
CodePaneView Property

Returns a value indicating whether the code pane is in Procedure view or Full Module view. Read-only.

**Return Values**

The CodePaneView property return values are:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbext_cv_ProcedureView</td>
<td>The specified code pane is in Procedure view.</td>
</tr>
<tr>
<td>Vbext_cv_FullModuleView</td>
<td>The specified project is in Full Module view.</td>
</tr>
</tbody>
</table>
Returns the **collection** that contains the object you are working with. Read-only.

**Remarks**

Most objects in this object model have either a **Parent** property or a **Collection** property that points to the object’s parent object.

Use the **Collection** property to access the **properties**, **methods**, and **controls** of the collection to which the object belongs.
Returns the **CommandBarEvents** object. Read-only.

**Settings**

The setting for the [argument](#) you pass to the **CommandBarEvents** property is:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbcontrol</td>
<td>Must be an object of type CommandBarControl.</td>
</tr>
</tbody>
</table>

**Remarks**

Use the **CommandBarEvents** property to return an [event source object](#) that triggers an event when a command bar button is clicked. The argument passed to the **CommandBarEvents** property is the command bar control for which the Click event will be triggered.
Returns a Long containing the number of items in a collection. Read-only.
CountOfDeclarationLines Property

Returns a `Long` containing the number of lines of code in the Declarations section of a code module. Read-only.
CountOfLines Property

Returns a `Long` containing the number of lines of code in a `code module`. Read-only.
Returns a **Long** containing the number of lines visible in a **code pane**. Read-only.
Description Property

Returns or sets a string expression containing a descriptive string associated with an object. For the VBProject object, read/write; for the Reference object, read-only.

Remarks

For the VBProject object, the Description property returns or sets a descriptive string associated with the active project.

For the Reference object, the Description property returns the descriptive name of the reference.
Designer Property

Returns the object that enables you to access the design characteristics of a component.

Remarks

If the object has an open designer, the Designer property returns the open designer; otherwise a new designer is created. The designer is a characteristic of certain VBComponent objects. For example, when you create certain types of VBComponent object, a designer is created along with the object. A component
can have only one designer, and it's always the same designer. The `Designer` property enables you to access a component-specific object. In some cases, such as in standard modules and class modules, a designer isn't created because that type of `VBComponent` object doesn't support a designer.

The `Designer` property returns `Nothing` if the `VBComponent` object doesn't have a designer.
FullPath Property

Returns a **String** containing the path and file name of the referenced **type library**. Read-only.
GUID Property

Returns a String containing the class identifier of an object. Read-only.
Returns a Boolean value indicating whether or not the VBComponent object has an open designer. Read-only.

Return Values

The HasOpenDesigner property returns these values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The VBComponent object has an open Design window.</td>
</tr>
<tr>
<td>False</td>
<td>The VBComponent object doesn't have an open Design window.</td>
</tr>
</tbody>
</table>
Returns or sets a **Single** containing the height of the window in **twips**. Read/write.

**Remarks**

Changing the **Height** property setting of a [linked window](#) or [docked window](#) has no effect as long as the window remains linked or docked.

**Important**  Objects, properties, and methods for controlling linked windows,
linked window frames, and docked windows are included on the Macintosh for compatibility with code written in Windows. However, these language elements will generate run-time errors when run on the Macintosh.
HelpContextID Property (VBA Add-In Object Model)

Returns or sets a String containing the context ID for a topic in a Microsoft Windows Help file. Read/write.
Returns or sets a `String` specifying the Microsoft Windows Help file for a `project`. Read/write.
IndexedValue Property (VBA Add-In Object Model)

Returns or sets a value for a member of a property that is an indexed list or an array.

Remarks

The value returned or set by the IndexedValue property is an expression that evaluates to a type that is accepted by the object. For a property that is an indexed list or array, you must use the IndexedValue property instead of the Value property. An indexed list is a numeric expression specifying index
IndexedValue accepts up to 4 indices. The number of indices accepted by IndexedValue is the value returned by the NumIndices property.

The IndexedValue property is used only if the value of the NumIndices property is greater than zero. Values in indexed lists are set or returned with a single index.
Returns a **Boolean** value indicating whether or not the **Reference** object points to a valid reference in the **registry**. Read-only.

**Return Values**

The **IsBroken** property returns these values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>True</strong></td>
<td>The <strong>Reference</strong> object no longer points to a valid reference in the registry.</td>
</tr>
<tr>
<td><strong>False</strong></td>
<td>The <strong>Reference</strong> object points to a valid reference in the registry.</td>
</tr>
</tbody>
</table>
Returns or sets a **Single** containing the location of the left edge of the window on the screen in **twips**. Read/write.

**Remarks**

The value returned by the **Left** property depends on whether or not the window is **linked** or **docked**.

**Note**  Changing the **Left** property setting of a linked or docked window has no
effect as long as the window remains linked or docked.

**Important** Objects, properties, and methods for controlling linked windows, linked window frames, and docked windows are included on the Macintosh for compatibility with code written in Windows. However, these language elements will generate run-time errors when run on the Macintosh.
Returns a string containing the specified number of lines of code.

**Syntax**

`object.Lines(startline, count) As String`

The `Lines` syntax has these parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**object**  Required. An [*object expression*](#) that evaluates to an object in the Applies To list.

**startline**  Required. A [*Long*](#) specifying the line number in which to start.

**count**  Required. A [*Long*](#) specifying the number of lines you want to return.

**Remarks**

The [*line numbers*](#) in a [*code module*](#) begin at 1.
LinkedWindowFrame Property

Returns the **Window** object representing the frame that contains the window. Read-only.

**Remarks**

The **LinkedWindowFrame** property enables you to access the object representing the linked window frame, which has properties distinct from the window or windows it contains. If the window isn't linked, the **LinkedWindowFrame** property returns **Nothing**.

**Important**  Objects, properties, and methods for controlling linked windows, linked window frames, and docked windows are included on the Macintosh for compatibility with code written in Windows. However, these language elements will generate run-time errors when run on the Macintosh.
Returns a `Window` object representing the main window of the Visual Basic development environment. Read-only.

**Remarks**

You can use the `Window` object returned by the `MainWindow` property to add or remove `docked windows`. You can also use the `Window` object returned by the `MainWindow` property to maximize, minimize, hide, or restore the main window of the Visual Basic development environment.

**Important**  Objects, properties, and methods for controlling linked windows, linked window frames, and docked windows are included on the Macintosh for compatibility with code written in Windows. However, these language elements will generate run-time errors when run on the Macintosh.
Major Property

Returns a Long containing the major version number of the referenced type library. Read-only.

Remarks

The number returned by the Major property corresponds to the major version number stored in the type library to which you have set the reference.
Minor Property

Returns a Long indicating the minor version number of the referenced type library. Read-only.

Remarks

The number returned by the Minor property corresponds to the minor version number stored in the type library to which you have set the reference.
Returns a value containing the mode of the specified project. Read-only.

**Return Values**

The **Mode** property return values are:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbext_vm_Run</td>
<td>The specified project is in run mode.</td>
</tr>
<tr>
<td>vbext_vm_Break</td>
<td>The specified project is in break mode.</td>
</tr>
<tr>
<td>vbext_vm_Design</td>
<td>The specified project is in design mode.</td>
</tr>
</tbody>
</table>
Name Property (VBA Add-In Object Model)

Returns or sets a String containing the name used in code to identify an object. For the VBProject object and the VBComponent object, read/write; for the
**Property** object and the **Reference** object, read-only.

**Remarks**

The following table describes how the **Name** property setting applies to different objects.

<table>
<thead>
<tr>
<th>Object</th>
<th>Result of Using Name Property Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>VBProject</td>
<td>Returns or sets the name of the active <strong>project</strong>.</td>
</tr>
<tr>
<td>VBComponent</td>
<td>Returns or sets the name of the component. An error occurs if you try to set the <strong>Name</strong> property to a name already being used or an invalid name.</td>
</tr>
<tr>
<td>Property</td>
<td>Returns the name of the property as it appears in the <strong>Property Browser</strong>. This is the value used to index the <strong>Properties collection</strong>. The name can't be set.</td>
</tr>
<tr>
<td>Reference</td>
<td>Returns the name of the reference in code. The name can't be set.</td>
</tr>
</tbody>
</table>

The default name for new objects is the type of object plus a unique integer. For example, the first new Form object is Form1, a new Form object is Form1, and the third TextBox control you create on a form is TextBox3.

An object's **Name** property must start with a letter and can be a maximum of 40 characters. It can include numbers and underline (_) characters but can't include punctuation or spaces. **Forms** and **modules** can't have the same name as another public object such as **Clipboard**, **Screen**, or **App**. Although the **Name** property setting can be a **keyword**, property name, or the name of another object, this can create conflicts in your code.
Returns the number of indices on the property returned by the Property object.

**Remarks**

The value of the NumIndices property can be an integer from 0 – 4. For most properties, NumIndices returns 0. Conventionally indexed properties return 1. Property arrays might return 2.
Returns or sets the value of an object returned by a property. Read/write.

**Remarks**

If a property returns an object, you must use the **Object** property to return or set the value of that object.
Returns the object or collection that contains another object or collection. Read-only.

**Remarks**

Most objects have either a Parent property or a Collection property that points to the object's parent object in this object model. The Collection property is used if the parent object is a collection.
Use the **Parent** property to access the **properties**, **methods**, and **controls** of an object's parent object.
Protection Property

Returns a value indicating the state of protection of a project. Read-only.

**Return Values**

The **Protection** property return values are:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbext_pp_locked</td>
<td>The specified project is locked.</td>
</tr>
<tr>
<td>vbext_pp_none</td>
<td>The specified project isn't protected.</td>
</tr>
</tbody>
</table>
Returns the `ReferencesEvents` object. Read-only.

**Settings**

The setting for the argument you pass to the `ReferencesEvents` property is:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vbproject</code></td>
<td>If <code>vbproject</code> points to <code>Nothing</code>, the object that is returned will supply events for the <code>References</code> collections of all <code>VBProject</code> objects.</td>
</tr>
</tbody>
</table>
objects in the **VBProjects** collection. If `vbproject` points to a valid **VBProject** object, the object that is returned will supply events for only the **References** collection for that **project**.

**Remarks**

The **ReferencesEvents** property takes an argument and returns an event source object. The **ReferencesEvents** object is the source for events that are triggered when references are added or removed.
Returns a Boolean value indicating whether or not the object was edited since the last time it was saved. Read/write.

**Return Values**

The Saved property returns these values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The object has not been edited since the last time it was saved.</td>
</tr>
<tr>
<td>False</td>
<td>The object has been edited since the last time it was saved.</td>
</tr>
</tbody>
</table>

**Remarks**

The SaveAs method sets the Saved property to True.
Note  If you set the Saved property to False in code, it returns False, and the object is marked as if it were edited since the last time it was saved.
Returns the selected component. Read-only.

**Remarks**

The `SelectedVBComponent` property returns the selected component in the **Project window**. If the selected item in the **Project** window isn't a component, `SelectedVBComponent` returns **Nothing**.
Top Property

Returns or sets a `Single` specifying the location of the top of the window on the screen in `twips`. Read/write.

**Remarks**

The value returned by the **Top** property depends on whether or not the window is `docked`, `linked`, or in docking view.

**Note**  Changing the **Top** property setting of a linked or docked window has no
effect as long as the window remains linked or docked.

**Important** Objects, properties, and methods for controlling linked windows, linked window frames, and docked windows are included on the Macintosh for compatibility with code written in Windows. However, these language elements will generate run-time errors when run on the Macintosh.
Returns a **Long** specifying the line number of the line at the top of the code pane or sets the line showing at the top of the code pane. Read/write.

**Remarks**

Use the **TopLine** property to return or set the line showing at the top of the code pane. For example, if you want line 25 to be the first line showing in a code pane, set the **TopLine** property to 25.

The **TopLine** property setting must be a positive number. If the **TopLine** property setting is greater than the actual number of lines in the code pane, the setting will be the last line in the code pane.
Type Property (VBA Add-In Object Model)
Returns a numeric or string value containing the type of object. Read-only.

**Return Values**

The **Type** property settings for the **Window** object are described in the following table:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbext_wt_CodeWindow</td>
<td>0</td>
<td><strong>Code</strong> window</td>
</tr>
<tr>
<td>vbext_wt_Designer</td>
<td>1</td>
<td><strong>Designer</strong></td>
</tr>
<tr>
<td>vbext_wt_Browser</td>
<td>2</td>
<td><strong>Object Browser</strong></td>
</tr>
<tr>
<td>vbext_wt_Immediate</td>
<td>5</td>
<td><strong>Immediate</strong> window</td>
</tr>
<tr>
<td>vbext_wt_ProjectWindow</td>
<td>6</td>
<td><strong>Project window</strong></td>
</tr>
<tr>
<td>vbext_wt_PropertyWindow</td>
<td>7</td>
<td><strong>Properties window</strong></td>
</tr>
<tr>
<td>vbext_wt_Find</td>
<td>8</td>
<td><strong>Find</strong> dialog box</td>
</tr>
<tr>
<td>vbext_wt_FindReplace</td>
<td>9</td>
<td><strong>Search and Replace</strong> dialog box</td>
</tr>
<tr>
<td>vbext_wt_LinkedWindowFrame</td>
<td>11</td>
<td><strong>Linked window frame</strong></td>
</tr>
<tr>
<td>vbext_wt_MainWindow</td>
<td>12</td>
<td>Main window</td>
</tr>
<tr>
<td>vbext_wt_Watch</td>
<td>3</td>
<td><strong>Watch</strong> window</td>
</tr>
<tr>
<td>vbext_wt_Locals</td>
<td>4</td>
<td><strong>Locals</strong> window</td>
</tr>
<tr>
<td>vbext_wt_Toolbox</td>
<td>10</td>
<td><strong>Toolbox</strong></td>
</tr>
<tr>
<td>vbext_wt_ToolWindow</td>
<td>15</td>
<td><strong>Tool</strong> window</td>
</tr>
</tbody>
</table>

**Important** Objects, properties, and methods for controlling linked windows, linked window frames, and docked windows are included on the Macintosh for compatibility with code written in Windows. However, these language elements will generate run-time errors when run on the Macintosh.
The **Type** property settings for the **VBComponent** object are described in the following table:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbext_ct_StdModule</td>
<td>1</td>
<td>Standard module</td>
</tr>
<tr>
<td>vbext_ct_ClassModule</td>
<td>2</td>
<td>Class module</td>
</tr>
<tr>
<td>vbext_ct_MSForm</td>
<td>3</td>
<td>Microsoft Form</td>
</tr>
<tr>
<td>vbext_ct_ActiveXDesigner</td>
<td>11</td>
<td>ActiveX Designer</td>
</tr>
<tr>
<td>vbext_ct_Document</td>
<td>100</td>
<td>Document Module</td>
</tr>
</tbody>
</table>

The **Type** property settings for the **Reference** object are described in the following table:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbext_rk_TypeLib</td>
<td>0</td>
<td>Type library</td>
</tr>
<tr>
<td>vbext_rk_Project</td>
<td>1</td>
<td>Project</td>
</tr>
</tbody>
</table>

The **Type** property settings for the **VBProject** object are described in the following table:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbext_pt_HostProject</td>
<td>100</td>
<td>Host Project</td>
</tr>
<tr>
<td>vbext_pt_Standalone</td>
<td>101</td>
<td>Standalone Project</td>
</tr>
</tbody>
</table>
Value Property

Returns or sets a **Variant** specifying the value of the **property**. Read/write.

**Remarks**

Because the **Value** property returns a **Variant**, you can access any property. To access a list, use the **IndexedValue** property.

If the property that the **Property** object represents is read/write, the **Value** property is read/write. If the property is read-only, attempting to set the **Value** property causes an error. If the property is write-only, attempting to return the **Value** property causes an error.

The **Value** property is the default property for the **Property** object.
VBE Property

Returns the root of the VBE object. Read-only.

Remarks

All objects have a VBE property that points to the root of the VBE object.
Returns a *String* containing the version of Visual Basic for Applications that the application is using. Read-only.

**Remarks**

The *Version* property value is a string beginning with one or two digits, a period, and two digits; the rest of the string is undefined and may contain text or numbers.
For the **Window** object, returns or sets a **Boolean** value that specifies the visibility of a window. Read/write. For the **CodePane** object, returns a **Boolean** value that indicates whether or not the **code pane** is visible in the window. Read-only.

**Return Values**

The **Visible** property returns the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>True</strong></td>
<td>(Default) Object is visible.</td>
</tr>
<tr>
<td><strong>False</strong></td>
<td>Object is hidden.</td>
</tr>
</tbody>
</table>
Returns or sets a **Single** containing the width of the window in **twips**. Read/write.

**Remarks**

Changing the **Width** property setting of a **linked window** or **docked window** has no effect as long as the window remains linked or docked.

**Important**  Objects, properties, and methods for controlling linked windows,
linked window frames, and docked windows are included on the Macintosh for compatibility with code written in Windows. However, these language elements will generate run-time errors when run on the Macintosh.
Window Property

Returns the window in which the code pane is displayed. Read-only.
Returns or sets a numeric value specifying the visual state of the window. Read/write.

**Settings**

The **WindowState** property returns or sets the following values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbext_ws_Normal</td>
<td>0</td>
<td>(Default) Normal</td>
</tr>
<tr>
<td>vbext_ws_Minimize</td>
<td>1</td>
<td>Minimized (minimized to an icon)</td>
</tr>
<tr>
<td>vbext_ws_Maximize</td>
<td>2</td>
<td>Maximized (enlarged to maximum size)</td>
</tr>
</tbody>
</table>
AddIns Collection

Returns a collection of add-ins registered for VBA.

Syntax

\[ \text{object}.\text{AddIns} \]

Remarks

The \textbf{AddIns} collection is accessed through the \textbf{VBE} object. Every add-in listed in the Add-In Manager in VBE has an object in the \textbf{AddIns} collection.