

## Quick Start

{ewc HLP95EN.DLL, DYNALINK, "See Also:"QuickStartC"}

The following is an overview of creating a Web page using the Microsoft [ActiveX Control Pad](#).

### ► **To create a Web page**

- 1** For each page, create or open an [HTML](#) file.  
By default, when you start the ActiveX Control Pad, it creates an HTML file for you.
- 2** Add content to the page by editing HTML, [inserting ActiveX controls into HTML](#), or [creating one or more HTML Layouts](#) that contain [controls](#).  
The ActiveX Control Pad saves each HTML Layout in a file with an .alx extension, which you insert into HTML. The HTML file incorporates the HTML Layout at run time.
- 3** Use the [Script Wizard](#) to assign actions to the controls you've added, or to add VBScript or JavaScript to HTML or HTML Layouts.
- 4** Preview the page using Internet Explorer, version 3.0.

## What the ActiveX Control Pad Does

{ewc HLP95EN.DLL, DYNALINK, "See Also":"WhatInternetStudioDoesC"}

With the ActiveX Control Pad, you can create interactive, multimedia Web sites and applications that go beyond the capabilities of standard HTML. You can create Web pages that combine HTML code, ActiveX controls, HTML Layouts, and VBScript or JavaScript.

- The ActiveX Control Pad uses an HTML file as the master container for each Web page you create. You can write and edit HTML directly using the HTML Source Editor.
- You can add a single ActiveX control, such as a **TextBox** or a **ScrollBar**, onto an HTML page using the ActiveX Control Editor. The ActiveX Control Editor lets you set properties for the control, then places an `<OBJECT>` tag into HTML at the insertion point.
- The ActiveX Control Pad introduces the concept of the HTML Layout to Web design. An HTML Layout is a WYSIWYG drawing board to which you can add multiple controls. You can draw controls in the precise sizes and locations you want, group and align them, and even put one control on top of another.

The ActiveX Control Pad saves each HTML Layout in a file format with an .alx extension. You insert the HTML Layout into HTML, which incorporates the HTML Layout at run time. You can use multiple HTML Layouts on a single Web page.

- Using the Script Wizard, you can assign events and actions to each of the controls you've added. You can also create custom scripts in VBScript or JavaScript. The Script Wizard then inserts the appropriate `<SCRIPT>` tag into HTML.

The result for each Web page is a single HTML file that, at run time, can display all of the elements described above.

# Who Does What: HTML, HTML Layouts, and Scripts

{ewc HLP95EN.DLL, DYNALINK, "See Also: "WhoDoesWhatHTML2DandScriptsC"}

The ActiveX Control Pad goes beyond simple HTML editing to extend the tools you can use to create Web pages. The following table describes the general uses for each of these tools.

| <b>Tool</b>            | <b>Use to</b>   |
|------------------------|---|
| HTML Source Editor     | <p>Write or edit HTML directly. The HTML Source Editor doesn't check syntax, so use caution when directly editing HTML.</p> <p>Each Web page you create using the ActiveX Control Pad begins with an HTML file. Any ActiveX <u>controls</u>, HTML Layouts, or <u>scripts</u> you create using the other ActiveX Control Pad tools are added as <code>&lt;OBJECT&gt;</code> and <code>&lt;SCRIPT&gt;</code> <u>tags</u> to the HTML file for that page.</p>  |
| ActiveX Control Editor | <p><u>Insert a single ActiveX control</u>, such as a <b>TextBox</b>, onto an HTML page, and edit that control's <u>properties</u>. You can insert any control registered on your computer, including third-party controls. For each control you insert, the ActiveX Control Editor adds an <code>&lt;OBJECT&gt;</code> tag to the HTML file.</p>  |
| HTML Layout Editor     | <p><u>Create an HTML Layout</u> with multiple controls. Use the HTML Layout Editor when you want to take a WYSIWYG approach to designing a Web page, and you want to specify precisely the location, size, transparency, and layering of controls.</p> <p>When you save an HTML Layout, the ActiveX Control Pad creates a file with an .alx extension that's separate from the HTML file for the page. You then <u>insert the HTML Layout into HTML</u>, which incorporates the layout at <u>run time</u>. You can add multiple HTML Layouts to a single Web page.</p> <p><b>Tip</b> You can reuse any saved HTML Layout across multiple HTML files. For example, you could design a navigation toolbar and save it as an HTML Layout, then insert that HTML Layout on all of your Web pages.</p> |
| Script Wizard          | <p>Assign actions or "add code" to controls, or add VBScript or JavaScript to HTML or an HTML Layout. The Script Wizard lets you work in <u>two different views</u>: List and Code.</p> <p>Use <u>List View</u> to assign simple actions to <u>events</u> — for example, to play a sound effect when the user clicks a certain image — or to set property values, using a "point-and-click" approach.</p> <p>Use <u>Code View</u> to write scripts directly in VBScript or JavaScript — when you want to</p>  |

utilize the power and full support of the  
scripting language.

## Working with HTML Layouts

{ewc HLP95EN.DLL, DYNALINK, "See Also":"WorkingwithHTMLLayoutsC"}

An HTML Layout is a WYSIWYG design area where you can place multiple controls precisely. The ActiveX Control Pad saves each HTML Layout in a file with an .alx extension, which you insert into HTML. The HTML file incorporates the HTML Layout at run time.

How you approach creating an HTML Layout depends on how you want to use it:



| <b>Approach</b>  | <b>Use to</b>   |
|--|---|
| <u>Create an HTML Layout for the current page.</u>           | Quickly create an HTML Layout and insert it into the current page at the cursor position.   |
| <u>Create an HTML Layout for future use.</u>                 | Create and save an HTML Layout, which you'll insert into one or more pages at another time. For example, you might design a navigation bar and save it as an HTML Layout, then insert it later into multiple pages. |
| <u>Insert an existing HTML Layout into the current page.</u> | Insert an HTML Layout that you've created previously into the current page at the cursor position.  |

## Create an HTML Layout for the Current Page

{ewc HLP95EN.DLL, DYNALINK, "See Also:."CreateanHTMLLayoutfortheCurrentPageC"}

Use this procedure to create an HTML Layout and insert it into the current page at the cursor position.

### ► **To create an HTML Layout for the current page**

- 1** In the HTML Source Editor, click the location on the HTML page where you want to insert the HTML Layout. You must click a location between the beginning and ending `<HTML>` tags for the page.
- 2** On the **Edit** menu, click **Insert HTML Layout**.
- 3** In the **File Name** box, type a name for the new HTML Layout, click **Open**, and then click **Yes** to create the file.  
The HTML Source Editor inserts an `<OBJECT>` tag for the new HTML Layout into the HTML page.  
The HTML Source Editor also adds an **HTML Layout** icon  in the margin to the left of the `<OBJECT>` tag.
- 4** Click  to start the HTML Layout Editor.
- 5** On the **View** menu, click **Properties**, then specify the default properties for the HTML Layout, such as the height, width, and background color.
- 6** On the toolbox, click a control and draw it on the HTML Layout.
- 7** Assign properties to the controls you've added to the HTML Layout.
- 8** Close the HTML Layout Editor and click **Yes** to save the changes to the HTML Layout.

## Create an HTML Layout for Future Use

{ewc HLP95EN.DLL, DYNALINK, "See Also:":"CreateanHTMLLayoutforFutureUseC"}

Use this procedure to create and save an HTML Layout, which you'll insert into one or more pages at another time.

► **To create an HTML Layout for future use**

- 1** On the **File** menu, click **New HTML Layout** to start the HTML Layout Editor.
- 2** On the **View** menu, click **Properties**, then specify the default properties for the HTML Layout, such as the height, width, and background color.
- 3** On the toolbox, click a control and draw it on the HTML Layout.
- 4** Assign properties to the controls you've added to the HTML Layout.
- 5** On the **File** menu, click **Save** to save the HTML Layout.

To use the new HTML Layout, insert it into an HTML file.

**Tip** You can use any saved HTML Layout across multiple HTML files. For example, you could design a navigation toolbar and save it as an HTML Layout, then insert that HTML Layout into all of your Web pages.

## Insert an ActiveX Control into HTML

{ewc HLP95EN.DLL, DYNALINK, "See Also":"InsertanActiveXControlintoHTMLC"}

Using the HTML Source Editor and the ActiveX Control Editor, you can add a single ActiveX control, such as a **ScrollBar**, at a specific place in HTML.


### ► To insert an ActiveX control into HTML

- 1 On the **File** menu, click **Open**, then choose the HTML file into which you'll insert the control.
- 2 Click the location in HTML where you want to insert the control. You must click a location between the beginning and ending `<HTML>` tags for the page.
- 3 On the **Edit** menu, click **Insert ActiveX Control**.
- 4 From the **Insert ActiveX Control** dialog box, select the control you want to add.

**Note** The **Insert ActiveX Control** dialog box lists all of the controls registered on your system, including any uncertified custom controls from third-party vendors. Use caution when inserting uncertified custom controls, as they may cause unpredictable results when your Web page is viewed.

- 5 Using the ActiveX Control Editor, draw the control and assign its properties.

When you close the ActiveX Control Editor, the ActiveX Control Pad inserts an `<OBJECT>` tag for that control into HTML.

**Tip** For each control you insert into HTML, the HTML Source Editor adds a **Control** icon  in the margin to the left of the `<OBJECT>` tag. To edit the control later, click



- 6 On the **File** menu, click **Save** to save your HTML file and the control you just inserted into it.




## Insert an Existing HTML Layout into the Current Page

{ewc HLP95EN.DLL, DYNALINK, "See Also":"InsertanExistingHTMLLayoutintotheCurrentPageC"}

Each HTML Layout is saved as a text file with an .alx extension, independent of HTML, so that you can use the same layout across multiple Web pages or combine multiple layouts on the same page. Use this procedure to insert an existing HTML Layout into the current page at the cursor position.

### ► **To insert an existing HTML Layout into the current page**

- 1** Create or open the HTML file for the Web page.
- 2** In the HTML Source Editor, click an insertion point anywhere between the `<BODY>` tags.
- 3** On the **Edit** menu, click **Insert HTML Layout**, then choose the HTML Layout that you want to insert.

When you click **Open**, the HTML Source Editor inserts an `<OBJECT>` tag referencing the .alx file in the HTML code. For each HTML Layout you insert into HTML, the HTML Source Editor adds an **HTML Layout** icon  in the margin to the left of the `<OBJECT>` tag. To edit the layout later, click



**Tip** If you're using multiple HTML Layouts on a single Web page, insert the layouts into the HTML code in the order that you want them to appear at run time.

## Editing Controls, HTML Layouts, and Scripts

{ewc HLP95EN.DLL, DYNALINK, "See Also:""EditingControlsHTMLLayoutsandScriptsC"}

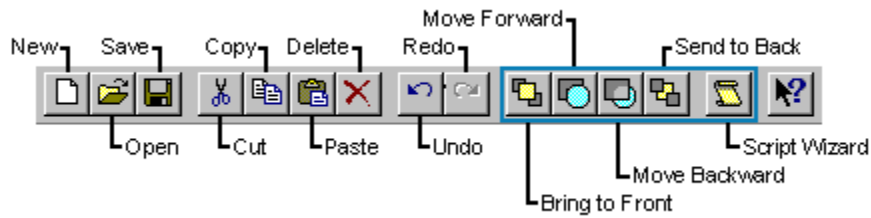
Whenever you insert an ActiveX control, an HTML Layout, or a script into HTML, the HTML Source Editor adds an icon in the margin to the left of that control, layout, or script. To edit the control, layout, or script, click the appropriate icon:

| <b>Click</b>  | <b>To edit the corresponding</b>  |
|---|-----------------------------------|
|  <b>Control</b> icon     | Control on that line of HTML.     |
|  <b>HTML Layout</b> icon | HTML Layout on that line of HTML. |
|  <b>Script</b> icon      | Script on that line of HTML.      |









## Working with the Toolbar

{ewc HLP95EN.DLL, DYNALINK, "See Also:":"WorkingwiththeToolbarC"}




The following table describes the buttons specific to the ActiveX Control Pad:

| Click   | To   |
|---|--|
|  | Bring the selected <u>control</u> to the top layer of the HTML Layout.   |
|  | Move the selected control up one layer on the HTML Layout.   |
|  | Move the selected control down one layer on the HTML Layout.   |
|  | Send the selected control to the bottom layer of the HTML Layout.  |
|  | <u>Start the Script Wizard to script events</u> on an HTML page or an HTML Layout.   |
|   | <b>Note</b> In the <b>HTML Source Editor</b> or the <b>HTML Layout Editor</b> , select the object you want to script before clicking  . |

## View or Edit Source Code for an HTML Layout

{ewc HLP95EN.DLL, DYNALINK, "See Also":"VieworEditSourceCodeforanHTMLLayoutC"}

### ► To view or edit source code for an HTML Layout

- 1 In the HTML Source Editor, click  to the left of the HTML Layout you want to view or edit.
- 2 In the HTML Layout Editor, using the right mouse button, click to display the shortcut menu, then click **View Source Code**.

The HTML Layout Editor closes, and the Windows Notepad opens, showing the source code for the HTML Layout.

- 3 View or edit the source code, then close Notepad when you're finished.

**Note** Use caution when editing source code for an HTML Layout. Neither Notepad nor the ActiveX Control Pad performs any syntax checking of source code that you edit. If you make any mistakes, you could corrupt the HTML Layout and potentially render it unusable.

## Choose List View or Code View

{ewc HLP95EN.DLL, DYNALINK, "See Also:"ChooseListVieworCodeViewC"}

The Script Wizard has two views: List view and Code view. You can switch between views by clicking the appropriate option at the bottom of the Script Wizard window. You can also specify the default view to use when the Script Wizard starts.


- Use **List view** to assign simple actions to events, or to set property values, using a “point-and-click” approach.
- Use **Code view** to write scripts directly in VBScript or JavaScript — when you want to use the power and full support of the scripting language.

► **To specify the default Script Wizard view**

- 1** On the **Tools** menu, point to **Options**, and then click **Script**.
- 2** In the **Script Pane View** area, select the default view you want to use.

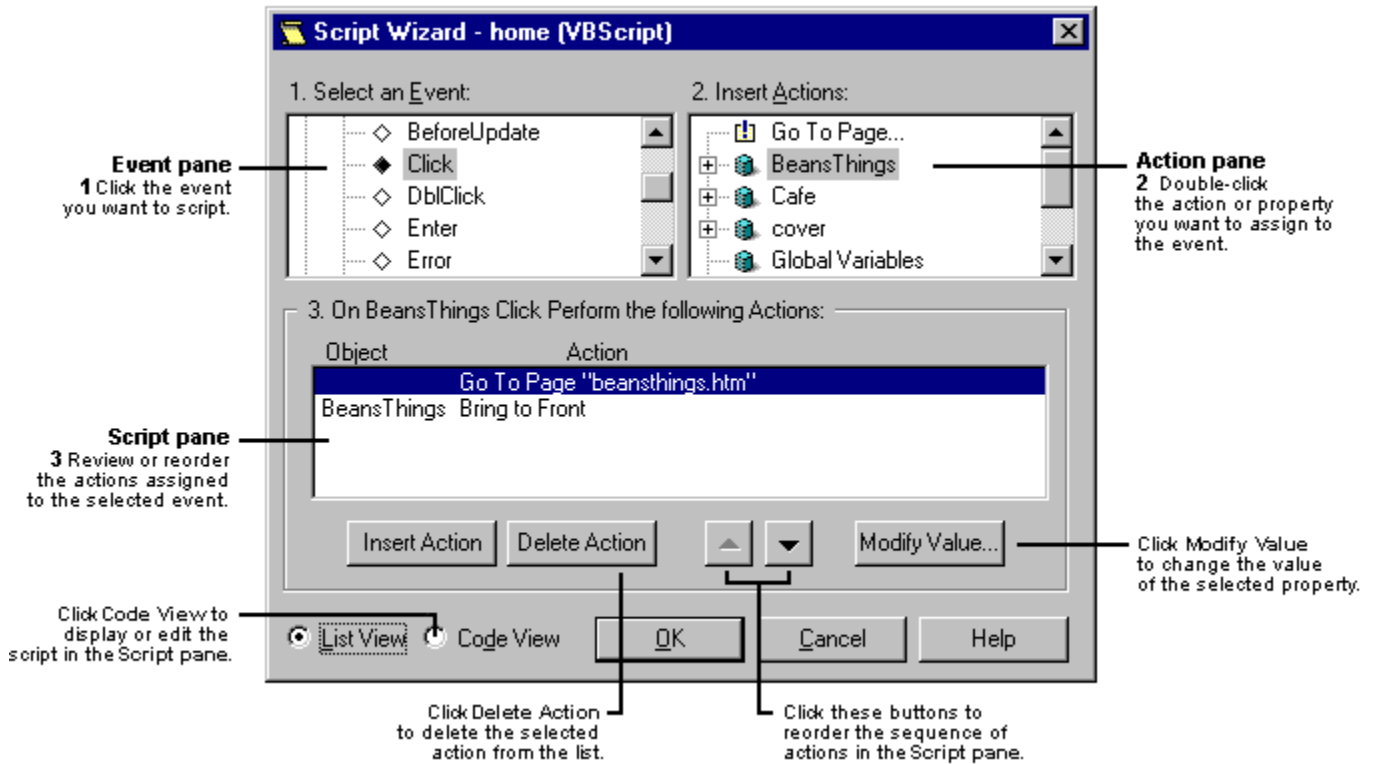
## Start the Script Wizard

{ewc HLP95EN.DLL, DYNALINK, "See Also:"StarttheScriptWizardC"}

- 1 Click the editor that contains the object you want to script.  
For example, to script an ActiveX control that you've inserted into HTML, click the **HTML Source Editor**.
- 2 On the **ActiveX Control Pad toolbar**, click .

# Using the Script Wizard in List View

{ewc HLP95EN.DLL, DYNALINK, "See Also:":"UsingtheScriptWizardinListViewC"}



## The Event Pane

The **Event pane** provides a hierarchical view of all the objects and events that you can script:

- If you started the Script Wizard from the **HTML Source Editor**, these include ActiveX controls that you've inserted into HTML and scriptable HTML tags. If you've inserted any HTML Layouts into HTML, they'll appear as HTML Layout controls.
- If you started the Script Wizard from the **HTML Layout Editor**, these include all of the controls in the layout.

In the hierarchy, objects are listed in alphabetical order by ID name. Under each object are the events that you can script. The icons represent different types of events and objects.

When you click an event, the Script Wizard displays that event handler in the **Script pane**.

## The Action Pane

The **Action pane** provides a hierarchical view of the actions and properties you can use in the event handler, as well as the global variables and procedures defined for the page. The icons represent different types of actions, properties, and objects.

- When you double-click an action, the Script Wizard adds that action to the list in the **Script pane**.
- When you double-click a property, you'll see a dialog box prompting you to choose a value for the property. The type of dialog box you'll see, and the values you can select, depend on the type of property you've double-clicked.

Once you've specified a value for the property, the Script Wizard adds that property to the list in the **Script pane**.

**Tip** You can quickly script a jump to another page, change a control's front to back layering, or hide or show a control, by double-clicking the **Go To Page...**, **Bring To Front/Send To Back**, or **Hide Control/Show Control** actions.

## The Script Pane

You can script multiple actions for any given event, and they'll be executed in the order they appear in the list in the **Script pane**. Use the **Up** and **Down Arrow** buttons to reorder the actions in the list, and the **Insert Action** and **Delete Action** buttons to add or remove actions from the list. If you specified a property, you can edit it by selecting that property and clicking the **Modify Value** button.

In the **Event pane**, if you click an event handler that's associated with a custom action — for example, a script that contains an “if” statement — you'll see a message in the **Script pane** advising you to click **Code view** to edit the action.

When you click **OK** or close the Script Wizard, the event handlers you create, and the global variables and procedures you define, are stored as VBScript or JavaScript in the HTML file for that page, or, for an HTML Layout, in the .alx file. To discard any changes you don't want to take effect, click **Cancel**.

## For More Information

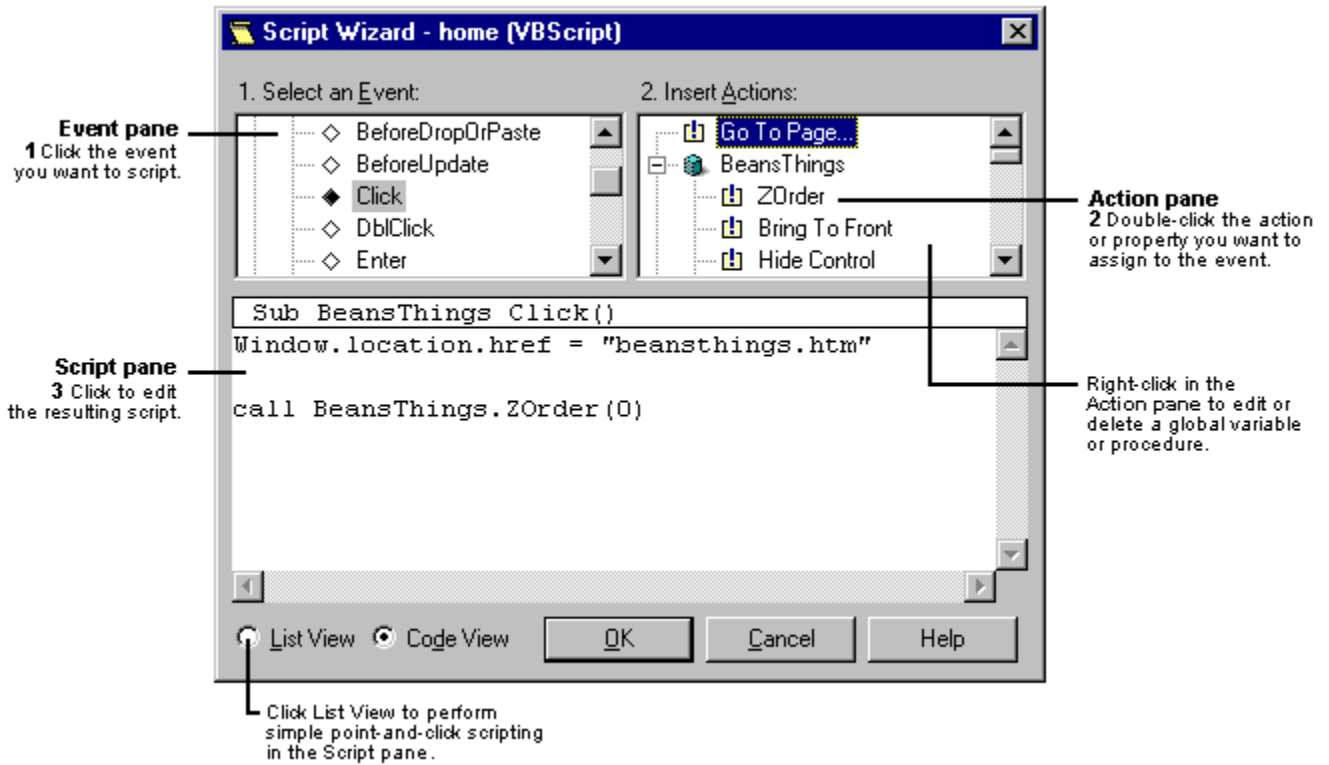
[HTML Reference](#)

[VBScript Reference](#)



# Using the Script Wizard in Code View

{ewc HLP95EN.DLL, DYNALINK, "See Also:":"UsingtheScriptWizardinCodeViewC"}



## The Event Pane

The **Event pane** provides a hierarchical view of all the objects and events that you can script:

- If you started the Script Wizard from the **HTML Source Editor**, these include ActiveX controls that you've inserted into HTML and scriptable HTML tags. If you've inserted any HTML Layouts into HTML, they'll appear as HTML Layout controls.
- If you started the Script Wizard from the **HTML Layout Editor**, these include all of the controls in the layout.

In the hierarchy, objects are listed in alphabetical order by ID name; under each object are the events that you can script. The icons represent different types of objects and events. When you click an event, the script for that event appears in the **Script pane**.

## The Action Pane

The **Action pane** provides a hierarchical view of all the actions, properties, variables, and procedures that can be invoked; each is represented by a different icon. When you double-click an action, the script for that action appears in the **Script pane**.

**Tip** You can quickly script a jump to another page, change a control's front to back layering, or hide or show a control, by double-clicking the **Go To Page...**, **Bring To Front/Send To Back**, or **Hide Control/Show Control** actions.

## The Script Pane

The **Script pane** displays the actual script in the default scripting language you specified for the page, either VBScript or JavaScript. Click an insertion point to edit the script. The Script Wizard

automatically adds **end sub** or **}** end-of-procedure marks to any script you create.

**Tip** In the **Script pane**, you can **Cut**, **Copy**, or **Paste** script, or change the display **Font**, using the shortcut menu.

When you click **OK** or close the Script Wizard, the scripts you create, and the global variables and procedures you define, are stored as VBScript or JavaScript in either the HTML file for that page, or, for an HTML Layout, in the .alx file. To discard any script or changes you don't want to take effect, click **Cancel**.

### **For More Information**







[HTML Reference](#)

[VBScript Reference](#)

## Event and Action Pane Icons

{ewc HLP95EN.DLL, DYNALINK, "See Also:"EventandActionPanelIconsC"}

In the Script Wizard, each item in the **Event** and **Action panes** is preceded by an icon. The following table describes each of these icons.

| <b>Icon</b>   | <b>Description</b>                  |
|---|-------------------------------------|
|  | An event that hasn't been scripted. |
|  | An event that has been scripted.    |
|  | An action.                          |
|  | A <u>property</u> .                 |
|  | An object.                          |
|  | The window object.                  |

## Create Event Handlers

{ewc HLP95EN.DLL, DYNALINK, "See Also":"CreateEventHandlersC"}

- 1** In the **Event pane** of the Script Wizard, click the Plus sign (+) next to the object you want to script. The hierarchy expands to display all of the events you can script for that specific object. ♦ indicates scripted events;  
♦ indicates events that haven't been scripted.
- 2** Click the event you want to script. For example, click **MouseOver** to specify what happens when the user moves the mouse pointer over the object.
- 3** In the **Action pane**, double-click the action, property, procedure, or variable you want to add to the event handler. For example, double-click **BackColor** to specify a background color that will change when the user moves the mouse pointer over the object.
  - If you're working in List view and you double-click a property, you'll see a dialog box asking you to choose a value for the property. The type of dialog box you'll see, and the values you can select, depend on the type of property you've double-clicked.
  - If you're working in Code view and you double-click a property or variable, edit the script in the **Script pane** and type a value for the property or variable.

## Delete Event Handlers

{ewc HLP95EN.DLL, DYNALINK, "See Also:"DeleteEventHandlersC"}

- 1 In the **Event pane** of the Script Wizard, click the Plus sign (+) next to the object that contains the event handler you want to delete.

The hierarchy expands to display all of the events for that specific object. ♦ indicates events that have been scripted.

- 2 Click the event.
- 3 Using the right mouse button, click to display the shortcut menu, then click **Delete Event Handler**.

## Define Global Variables

{ewc HLP95EN.DLL, DYNALINK, "See Also:":"DefineGlobalVariablesC"}

- 1 With the pointer in the **Action pane** of the Script Wizard, using the right mouse button, click to display the shortcut menu, then click **New Global Variable**.
- 2 In the **New Global Variable** dialog box, type the name of the global variable you want to add to the page.

You can include subscripts in VBScript or an initial value in JavaScript, in the proper syntax for that scripting language.

When you click **OK**, the Script Wizard inserts that variable at the beginning of the HTML file for that page, or, for an HTML Layout, in the .alx file, in the form **dim** *variable-name* for VBScript or **var** *variable-name* for JavaScript.

## Edit or Delete Global Variables

{ewc HLP95EN.DLL, DYNALINK, "See Also:"EditorDeleteGlobalVariablesC"}

- 1 In the **Action pane** of the Script Wizard, select the global variable you want to edit or delete.  
**Note** In JavaScript, only variables defined with the **var** statement are considered global variables.
- 2 Using the right mouse button, click to display the shortcut menu, then click **Edit** or **Delete**.  
**Note** You can't delete a global variable if it's part of a multivariable **dim** or **var** statement.

## Define Procedures

{ewc HLP95EN.DLL, DYNALINK, "See Also:":"DefineProceduresC"}

- 1 With the pointer in the **Action pane** of the Script Wizard, using the right mouse button, click to display the shortcut menu, then click **New Procedure**.
- 2 In the **Script pane**, edit the procedure:
  - If you're working in **List view**, in the **Action pane** double-click the action or property to add to the procedure.
  - If you're working in **Code view**, in the **Script pane** edit the script for the new procedure.

When you click **OK**, the Script Wizard inserts the procedure at the beginning of the HTML file for that page, or, for an HTML Layout, in the .alx file.



## Specify the Scripting language

{ewc HLP95EN.DLL, DYNALINK, "See Also:":"SpecifytheScriptLanguageC"}

The Script Wizard lets you create scripts in either VBScript or JavaScript. However, you can use only one scripting language per page, and you can't change the scripting language for a page that already contains script.



### To specify the scripting language

- 1 On the **Tools** menu, point to **Options**, and then click **Script**.
- 2 In the **Default Script Language** area, select the scripting language you want to use.

## Specify the Script Pane Font

{ewc HLP95EN.DLL, DYNALINK, "See Also:"SpecifytheScriptPaneFontC"}

- 1 With the pointer in the **Script pane**, in Code view, using the right mouse button, click to display the shortcut menu, then click **Font**.
- 2 Select the font you want to use in the **Script pane**.

## Script a Jump to Another Page

```
{ewc HLP95EN.DLL, DYNALINK, "See Also:":"ScriptaJumpToAnotherPageC"}
```

Use this procedure to script an event that jumps to another page.



### To script a jump to another page

- 1 In the **Event pane** of the Script Wizard, click the Plus sign (+) next to the object you want to script. The hierarchy expands to display all of the events for that specific object.
- 2 Click the event from which you want to create a jump. For example, click **Click** to jump to another page when the user clicks the object.
- 3 In the **Action pane**, double-click the **Go To Page...** action.
  - If you're working in List view, in the **Go To Page** dialog box, type the URL, without beginning or ending quotation marks, for the jump destination page, then click **OK**.
  - If you're working in Code view, in the **Script pane**, type the URL of the jump destination page for the **window.location.href = ""** property.

## Hide or Show a Control

{ewc HLP95EN.DLL, DYNALINK, "See Also:":"HideorShowaControlC"}

Use this procedure to script an event that hides or shows a control.



### To hide or show a control

- 1 In the **Event pane** of the Script Wizard, click the Plus sign (+) next to the object that contains the event you want to script. The hierarchy expands to display all of the events for that specific object.
- 2 Click the event that will initiate the hide or show action. For example, click **Click** to hide or show a control when the user clicks the object in the **Event pane**.
- 3 In the **Action pane**, click the Plus sign (+) next to the control that you want to hide or show. The hierarchy expands to display all of the actions and properties for that specific control.
- 4 Double-click the **Hide Control** or **Show Control** action as appropriate.

## Change the Front/Back Order of a Control

{ewc HLP95EN.DLL, DYNALINK, "See Also:":"ChangetheFrontBackOrderofaControlC"}

Use this procedure to script an event that moves a control to the front or back layer, or z-order, of an HTML Layout.



### To change the front/back order of a control

- 1 In the **Event pane** of the Script Wizard, click the Plus sign (+) next to the object that contains the event you want to script. The hierarchy expands to display all of the events for that specific object.
- 2 Click the event you want to initiate the front/back reordering action. For example, click **Click** to bring a control to the front of the z-order when the user clicks the object in the **Event pane**.
- 3 In the **Action pane**, click the Plus sign (+) next to the control that you want to reorder front or back. The hierarchy expands to display all of the actions and properties for that specific control.
- 4 Double-click the **Bring to Front** or **Send to Back** action as appropriate.

# Script Wizard Technical Notes

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ScriptWizardTechnicalNotesC"}

## Differences between List view and Code view

In **List view**, you can create event handlers that use these simple actions:

- Any method invocation on any object or procedure that takes no arguments.
- Any method invocation on any object or procedure that takes the same number and names of arguments as the event handler.
- The **Go To Page**, **Hide** or **Show Control**, or **Bring to Front/Send to Back** actions.
- Any assignment to an object property or global variable.

In **Code view**, on the other hand, you have full support for the default scripting language, including control flow.

## Scriptable HTML elements

The ActiveX Control Pad doesn't support scripting of HTML `<A HREF="...">` or `<FRAMESET>` tags.

## Scripting the Window object

In an HTML Layout .alx file, you can script only the **window.location.href** property for the Window object. In an HTML Source Editor page, the entire Window object is available.

## <INPUT> elements with common NAME=

Option buttons and other `<INPUT>` elements that share the same `NAME=` are not differentiated from one another in the Script Wizard **Event pane**.

For example, in the case `<INPUT TYPE=RADIO>`, each item is displayed in the following way:

```
NAME 'VALUE'
```

So if you have three option buttons, all named **Color**, but each having a distinct value, they might appear as:

- Color 'Blue'
- Color 'Red'
- Color 'Green'

## Reading HTML

When you start the Script Wizard, it reads the HTML file and searches for any occurrences of script. The first language found is established as the default scripting language for that page.

If the Script Wizard finds more than one variable on a **dim** or **var** line, that variable can't be deleted or modified.

The ActiveX Control Pad does not support the `SRC=` attribute of the HTML `<SCRIPT>` tag. If your HTML page includes such a tag, you'll see a warning message when you start the Script Wizard.

## Writing HTML

When you close the Script Wizard, existing scripts are saved back into the HTML file in the same location. The ActiveX Control Pad adds HTML comment tags `<!-- ... >` around any new script, and maintains them (if they were already there) on any existing script.

All new external event handlers are placed before, and as near as possible to, the object within the

<BODY> tags of the HTML file, or, for an .alx file, before the <DIV> tag.

Any global variables or user-defined procedures will appear in <SCRIPT> tags within the <HEAD> section of the HTML file, or, for an .alx file, before any event handlers.

If the Script Wizard inserts a new <SCRIPT> tag before any others in the HTML file, the new <SCRIPT> tag will specify the default scripting language.

### **Quotation marks in scripts**

External **List view** scripts use double quotes ("), while internal **List view** scripts use single quotes (').

### **For More Information**

[HTML Reference](#)

[VBScript Reference](#)

# HTML Layout Control

```
{ewc HLP95EN.DLL, DYNALINK, "See Also": "isObjHTMLLayoutControlC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "isObjHTMLLayoutControlX ":1} {ewc HLP95EN.DLL, DYNALINK, "Properties": "isObjHTMLLayoutControlIP"} {ewc HLP95EN.DLL, DYNALINK, "Methods": "isObjHTMLLayoutControlIM"} {ewc HLP95EN.DLL, DYNALINK, "Events": "isObjHTMLLayoutControlE"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "isObjHTMLLayoutControlS "}
```

References an HTML Layout and renders it at run time.

## Remarks

An HTML Layout is a WYSIWYG drawing board to which you can add multiple controls. You can draw controls in the precise sizes and locations you want, group and align them, and even put one control on top of another.

The ActiveX Control Pad saves each HTML Layout in a file format with an .alx extension. When you insert an HTML Layout into HTML, the ActiveX Control Pad adds an **HTML Layout** control for each layout that you insert. The **HTML Layout** control is what actually renders the HTML Layout at run time.



## HTML Layout

```
{ewc HLP95EN.DLL, DYNALINK, "See Also": "isObjActiveXLayoutC"}      {ewc HLP95EN.DLL, DYNALINK,
"Example": "isObjActiveXLayoutX ":1}      {ewc HLP95EN.DLL, DYNALINK, "Properties": "isObjActiveXLayoutP"}
{ewc HLP95EN.DLL, DYNALINK, "Methods": "isObjActiveXLayoutM"}      {ewc HLP95EN.DLL, DYNALINK,
"Events": "isObjActiveXLayoutE"}      {ewc HLP95EN.DLL, DYNALINK, "Specifics": "isObjActiveXLayoutS "}
```

A WYSIWYG design area where you can place multiple controls precisely.

### Remarks

An HTML Layout is a WYSIWYG drawing board to which you can add multiple controls. Using the HTML Layout Editor, you can draw controls in the precise sizes and locations you want, group and align them, and even put one control on top of another. You can also specify properties for an HTML Layout, such as the background color, in the HTML Layout Editor's Properties window.

The ActiveX Control Pad saves each HTML Layout in a file format with an .alx extension. When you insert an HTML Layout into HTML, the ActiveX Control Pad adds an **HTML Layout** control for each layout that you insert. The **HTML Layout** control is what actually renders the HTML Layout at run time.

If you insert multiple HTML Layouts into HTML, the layouts will be rendered at run time in the order they appear in HTML.

## HotSpot Control

```
{ewc HLP95EN.DLL, DYNALINK, "See Also": "isObjHotSpotC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "isObjHotSpotX": 1} {ewc HLP95EN.DLL, DYNALINK, "Properties": "isObjHotSpotP"} {ewc HLP95EN.DLL, DYNALINK, "Methods": "isObjHotSpotM"} {ewc HLP95EN.DLL, DYNALINK, "Events": "isObjHotSpotE "} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "isObjHotSpotS "}
```

Specifies a region for exposing events.

### Remarks

Assigning actions to the **HotSpot** control's **MouseEnter** and **MouseExit** events determines what happens when a user moves the mouse pointer over the **HotSpot**.

You can also use **HotSpot** as an alternative to image maps in HTML. To do this, place multiple **HotSpot** controls over an **Image** control and assign actions to the **HotSpot** controls' events. By default, the **HotSpot** is invisible at run time because the default value of the **BackStyle** property is **Transparent** and the **BorderStyle** property is **None**.

To make your HTML Layout more accessible to keyboard-only users, assign actions to the **HotSpot** control's **Enter** event and make sure that the **Enabled** property is set to **True**. At run time, keyboard-only users will then be able to tab to the **HotSpot** and press ENTER to trigger the event.

The default event for a **HotSpot** is the Click event.

# Image Control

```
{ewc HLP95EN.DLL, DYNALINK, "See Also": "isObjImageC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "isObjImageX ":1} {ewc HLP95EN.DLL, DYNALINK, "Properties": "isObjImageP "} {ewc HLP95EN.DLL, DYNALINK, "Methods": "isObjImageM "} {ewc HLP95EN.DLL, DYNALINK, "Events": "isObjImageE "} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "isObjImageS "}
```

Displays a picture.

## Remarks

The **Image** control lets you crop, size, or zoom a picture, but does not allow you to edit the contents of the picture. For example, you can't use **Image** to change the colors in the picture, to make the picture transparent, or to refine the image of the picture. You must use image editing software for these purposes.

**Image** supports the following formats:

- .gif ('87 and '89)
- .jpg
- .wmf
- .bmp

**Note** The picture is not actually embedded into the control. The control references the picture at run time based on the URL specified by the **PicturePath** property

The default event for **Image** is the Click event.

## CodeBase Property

{ewc HLP95EN.DLL, DYNALINK, "See Also:" isProCodeBaseC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"isProCodeBaseX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"isProCodeBaseA"}  
HLP95EN.DLL, DYNALINK, "Specifics":"isProCodeBaseS"} {ewc

Specifies the URL of a control's COM object.

### Remarks

The **CodeBase** property makes it possible to automatically download ActiveX controls from a server to a user's machine.

The CodeBase property supports the following file types:

| <u>File type</u>                   | <u>Description</u>   |
|------------------------------------|--|
| PE<br>(portable executable)        | The PE (for example, .ocx, .dll, .exe) is downloaded, installed, and registered automatically if the control is not already registered on the user's computer. This is the simplest way to package a single-file ActiveX control, but it does not use file compression and isn't platform independent except with HTTP.                |
| .cab<br>(cabinet)                  | The .cab file contains one or more files, all of which are downloaded together in a single compressed cabinet file. One file in the cabinet is an .inf file providing further installation information. The .inf file may refer to files in the .cab as well as to files at other URLs.  |
| .inf<br>(installation information) | The stand-alone .inf file specifies various files that need to be downloaded and set up for an .ocx to run. The syntax of the .inf file supports URLs pointing to files to download as well as platform independence (by enumerating files for various platforms). This mechanism provides platform independence for non-HTTP servers. |

For specifics about creating PE, .cab, and .inf files and for the latest information about Internet Component Download, go to

<http://www.microsoft.com/intdev/signcode/codedwld.htm>  
on the Internet.

**Note** The **CodeBase** property can be set only at design time. It can't be set at run time.

## DrawBuffer Property

{ewc HLP95EN.DLL, DYNALINK, "See Also:." isProDrawBufferC"} {ewc HLP95EN.DLL, DYNALINK, "Example:."isProDrawBufferX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To:."isProDrawBufferA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics:."isProDrawBufferS"}

Specifies the suggested number of pixels set aside for off-screen memory in rendering an HTML Layout.

### Syntax

object.**DrawBuffer** [= *value*]

| Part          | Description   |
|---------------|---|
| <i>object</i> | Required. A valid object name.  |
| <i>value</i>  | An integer between 16,000 to 1,048,576 equal to the maximum number of pixels the object will render off-screen. The default value is 32,000, which covers, for example, an area of 80x400 pixels. |

### Remarks

The **DrawBuffer** property specifies the maximum number of pixels that can be drawn at one time as the display repaints. The actual memory used by the HTML Layout depends on the screen resolution of the display. If you set a large value for **DrawBuffer**, performance will be slower. A large buffer helps when several large images overlap.

The **DrawBuffer** property cannot be set from the Properties window in your .alx file. You can set **DrawBuffer** in Script Wizard by selecting the HTML Layout **onLoad** event.

## PicturePath Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "isProPicturePathC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "isProPicturePathX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "isProPicturePathA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "isProPicturePathS"}

Specifies the URL of the picture to display on **Image** control.

### Syntax

*object*.**PicturePath** = *URL*

The **PicturePath** property syntax has these parts:

| <b>Part</b>   | <b>Description</b>                   |
|---------------|--------------------------------------|
| <i>object</i> | Required. A valid object.            |
| <i>URL</i>    | Required. The URL of a picture file. |

### Remarks

**PicturePath** requires a complete URL. It does not support a UNC path.

## MouseEnter, MouseExit Events

```
{ewc HLP95EN.DLL, DYNALINK, "See Also:" isEvtMouseEnterMouseExitC"}          {ewc HLP95EN.DLL, DYNALINK,  
"Example":"isEvtMouseEnterMouseExitX":1}          {ewc HLP95EN.DLL, DYNALINK, "Applies  
To":"isEvtMouseEnterMouseExitA"}          {ewc HLP95EN.DLL, DYNALINK, "Specifics":"isEvtMouseEnterMouseExitS"}
```

MouseEnter occurs when the mouse pointer is moved over the control. MouseExit occurs when the mouse pointer is moved off of the control.

### Syntax

**Private Sub** *object*\_MouseEnter( )

**Private Sub** *object*\_MouseExit( *Cancel As Boolean*)

The **MouseEnter** and **MouseExit** event syntax has these parts:

| <b>Part</b>   | <b>Description</b>  |
|---------------|---|
| <i>object</i> | Required. A valid object name.  |
| <i>Cancel</i> | Required. Event status. <b>False</b> indicates that the control should handle the event (default). <b>True</b> indicates that the application should handle the event and the focus should remain at the current control. |

### Remarks

You can use the MouseEnter and MouseExit events to make something interesting happen, like playing sound files, when the mouse pointer hovers over an object.

## onLoad Event

```
{ewc HLP95EN.DLL, DYNALINK, "See Also:." isEvtonLoadC"} {ewc HLP95EN.DLL, DYNALINK, "Example"."isEvtonLoadX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To"."isEvtonLoadA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics"."isEvtonLoadS"}
```

OnLoad occurs when the entire contents of the HTML Layout (.ALX file) are created and before the Window onLoad event occurs.

### Syntax

**sub** *object.onLoad=function-name*

| Part | Description |
|------|-------------|
|------|-------------|

|               |   |
|---------------|---|
| <i>object</i> | Required. The filename of the HTML Layout object (.ALX file). |
|---------------|---|

|                      |   |
|----------------------|---|
| <i>function-name</i> | An object expression which evaluates to a scripting function. |
|----------------------|---|

### Remarks

The order of onLoad events is the HTML Layout, then the Window onLoad event. The Window onLoad event only fires after the entire contents of the window have been rendered.

You can script the Window onLoad event from HTML but not from within the .ALX. Therefore, when editing an .ALX file, the Script Wizard does not display the Window Load/Unload events.

There is no HTML Layout onUnload event.



## Insert ActiveX Control Dialog Box

{ewc HLP95EN.DLL, DYNALINK, "See Also:":"InsertActiveXControlDialogBoxC"}

This dialog box appears when you click **Insert ActiveX Control** on the **Edit** menu. Use it to select an ActiveX control, such as a **TextBox**, to insert into HTML.

In the Object Type list, click the control you want to insert, then click OK.

**Note** The Object Type list shows all of the controls registered on your system, including any uncertified custom controls from third-party vendors. Use caution when inserting uncertified custom controls, as they may cause unpredictable results when your Web page is viewed.

## Script Wizard Options Dialog Box

{ewc HLP95EN.DLL, DYNALINK, "See Also:"ScriptWizardOptionsDialogBoxC"}

This dialog box appears when, in the ActiveX Control Pad, you click the **Tools** menu, point to **Options**, and then click **Script**. Use this dialog box to set the default view, script language, and font for the Script Wizard.

| <b>To</b>   | <b>Click</b>   |
|---|--|
| Set the default view for the Script Wizard                              | <b>List View</b> or <b>Code View</b>                             |
| Set the default script language for the Script Wizard                   | <b>Visual Basic Script</b> or <b>JavaScript</b>                  |
| Change the font used in the Script Wizard Script pane when in Code view | <b>Script Pane Font</b> to display the Font Selection dialog box |

## HTML Layout Options Dialog Box

{ewc HLP95EN.DLL, DYNALINK, "See Also":"HTMLLayoutOptionsDialogBoxC"}

This dialog box appears when you click the **Tools** menu, point to **Options**, and then click **HTML Layout**. Use this dialog box to specify the default grid settings for the HTML Layout Editor.

| <b>To</b>  | <b>Do this</b>  |
|--|---|
| Change the size of the layout grid   | Under <b>Grid Settings</b> , adjust the number of points for <b>Vertical Spacing</b> or <b>Horizontal Spacing</b> . |
| Hide the grid  | Clear the <b>Show Grid</b> check box.   |
| Show the grid  | Select the <b>Show Grid</b> check box.  |
| Disable the grid so you can draw objects of any size in any location                                   | Clear the <b>Snap To Grid</b> check box.  |
| Enable the grid so you can draw objects that are automatically aligned to the nearest grid coordinates | Select the <b>Snap To Grid</b> check box.   |

## New Global Variable Dialog Box

{ewc HLP95EN.DLL, DYNALINK, "See Also":"NewGlobalVariableDialogBoxC"}


This dialog box appears when you're working in the Action pane of the Script Wizard, and you click **New Global Variable** on the shortcut menu. Use this dialog box to add a new global variable to HTML for the active Web page.

Type the name of the global variable you want to add. You can include subscripts or an initial value in the proper syntax for the script language you're using (either VBScript or JavaScript).

When you click OK, the ActiveX Control Pad then inserts that variable at the beginning of HTML, in the form **dim** *variable-name* for VBScript or **var** *variable-name* for JavaScript.

## New Dialog Box

{ewc HLP95EN.DLL, DYNALINK, "See Also":"NewDialogBoxC"}

This dialog box appears when you click  on the ActiveX Control Pad toolbar. Use this dialog box to create a new Web page or a new HTML Layout.

| <b>To</b>                                     | <b>Click</b>             |
|---|--------------------------|
| Create an <u>HTML</u> file for a new Web page | Internet Document (HTML) |
| Create an HTML Layout for future use.         | HTML Layout              |

**Note** You must insert the HTML Layout into one or more HTML pages at another time.

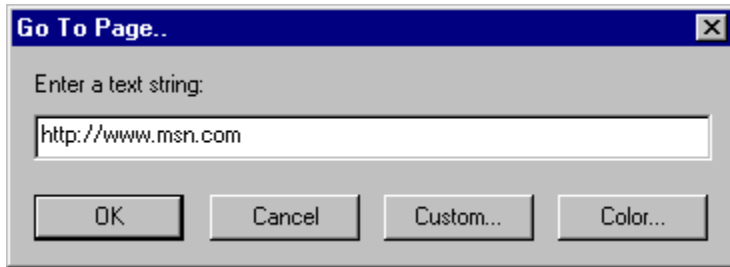
## Value Dialog Boxes

{ewc HLP95EN.DLL, DYNALINK, "See Also:ValueDialogBoxesC"}

In **List View**, the appropriate properties and global variables for events are displayed in the Action pane and can be assigned values.

Double-click the name of the action you wish to insert. A type of Value dialog box appears, depending on the allowable values for that particular action. Values consist of text strings, custom or variable names, numbers, Boolean values, or enumerations.

### Text String Value dialog box



This example of a Text String Value dialog box asks you to insert a text string for the GoTo Page.. action. You can either type in a known value, such as the URL for the page associated with this particular event, or click the Custom or Color buttons to assign a value.

When the value is inserted into the script, quotation marks are automatically added to the string. For example:

| <u>What you enter:</u> | <u>Result:</u> | <u>Result for INPUT, SELECT, TEXTAREA, or FORM event</u> |
|------------------------|----------------|--|
| Sample Text            | “Sample Text”  | ‘Sample Text’  |

You receive an error message if you enter an invalid text string. This can happen because of a single quotation mark (‘) or a double quotation mark (“) in the value you typed. For example:

| <u>What you enter:</u> | <u>Result:</u>                      | <u>Result for INPUT, SELECT, TEXTAREA, or FORM event</u> |
|------------------------|-------------------------------------|--|
| It’s the Internet      | “It’s the Internet”<br>(Valid)      | ‘It’s the Internet’<br>(Invalid)                         |
| It is the “Internet”   | “It is the “Internet””<br>(Invalid) | ‘It is the “Internet”’<br>(Valid)                        |

If you are scripting an INPUT, TEXTAREA, SELECT or FORM element, then the problem is the presence of a single quotation mark in your value. In all other cases, the problem is the presence of a double quotation mark in your value. The solution to this problem depends on the scripting language you are working with.

### VBScript Solution

If you are using VBScript, the solution is to double the number of quotation marks. If you are typing a double quotation mark in your text string which is causing problems, turn it into two (") characters: ("""). If you are typing a single quotation mark which is causing problems, turn it into two (') characters: ('?').

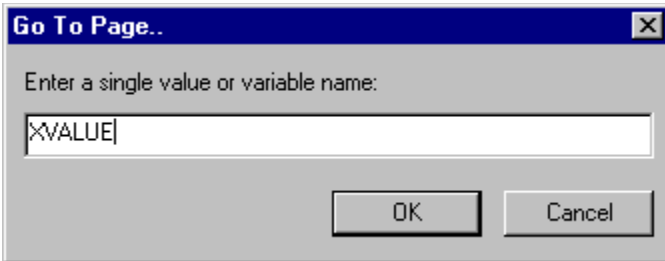
| <u>Invalid result:</u> | <u>What you should type:</u> |
|------------------------|------------------------------|
| ‘It’s the Internet     | It”s the Internet            |
| “It is the “Internet”” | It is the ""Internet""       |

### JavaScript Solution

If you are using JavaScript, you can place a backslash character in front of the problem character. For example, if a double quotation mark is causing problems, change it to (\"). Similarly, if a single quotation mark is causing problems, change it to (\').

|                        |                              |
|------------------------|------------------------------|
| <b>Invalid result:</b> | <b>What you should type:</b> |
| 'It's the Internet'    | It\'s the Internet           |
| "It is the 'Internet'" | It is the \'Internet\'       |

### Custom Value dialog box



This dialog box appears when clicking Custom from the previous Go To Page.. Text String Value dialog box. If you have a pre-defined variable, property value or value that is not a simple constant, you can enter it here.

The Custom Value dialog box is the default Value dialog box and appears when the selected property is not categorized as a text string, number, Boolean value, or enumeration. You can also bring this dialog box up by clicking the Custom option in one of the Value dialog boxes as previously displayed, or click the Modify Value option in the Script pane.

### Color dialog box

Clicking the Color button displays the standard Windows color picker window. Selecting a color creates a value in the Script pane depending on which property you are modifying.

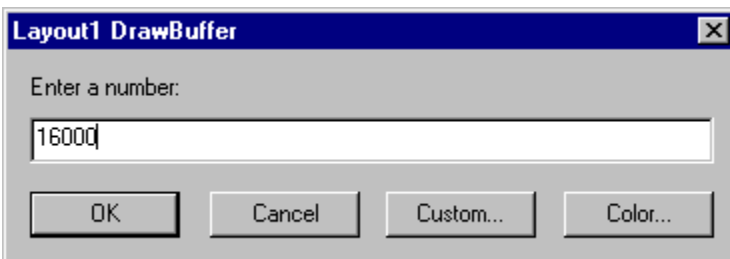
If you are modifying a Window object property, the value is displayed according to HTML rules in RRGGBB format as hex constants with the # symbol as the prefix.

If you are modifying a control property and reaching the Color dialog box directly or by choosing Color from the Number Value dialog box, the value is interpreted as an OLE color in BBGGRR format as hex constants, with the &H symbol as the prefix for VBScript, and the 0x prefix for JavaScript.

Note that the Color button is available for many properties and not just for changing the foreground or background colors of an object. For example, if you change the value of the **Caption** property from text such as "Push" to the color red, the actual text "#FF0000" is placed in your control rather than the color red.

HTML-named colors such as "red" or "white" are not supported values.

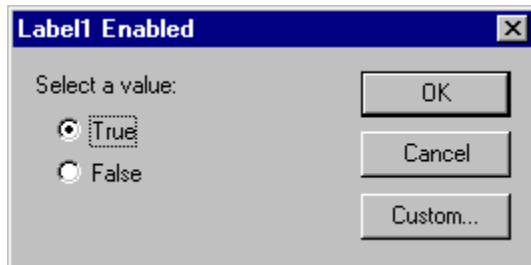
### Number Value dialog box



This example of a Number Value dialog box asks you to insert a number for the Layout1 **DrawBuffer** property. Only valid decimal, hexadecimal, or octal numbers in the syntax of the current scripting

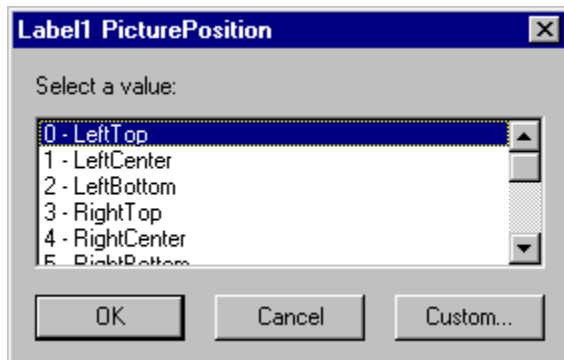
language may be entered in this dialog box. If an invalid number is entered, an error message appears. The Custom button, from which you can assign a single constant value or a variable assignment, is also available from this dialog box.

### Boolean Value dialog box



This example of a Boolean Value dialog box provides option buttons from which to choose the Boolean values **True** or **False** for the Label1 **Enabled** property. The Custom button, from which you can assign a single constant value or a variable assignment, is also available from this dialog box.

### Enumeration Value dialog box



This example of an Enumeration Value dialog box provides a list box of integer values from which to choose the Label1 **PicturePosition** property value. The Custom button, from which you can assign a single constant value or a variable assignment, is also available from this dialog box.



# Glossary

## **A-C**

accelerator key

ActiveX Control Pad

ANSI character set

argument

array

background color

browser

class

class identifier

clear

client region

collection

COM object

container

context ID

control

control group

cycle

## **D-F**

data format

default

design time

dominant control

drop source

event

event handler

focus

foreground color

function

## **G-M**

GIF

global variable

grid block

home page

HTML

HTTP

hyperlink

hypertext

IME

inherited property

Internet

intranet

JPEG

keyboard state

mask

method

modal

module

## **N-R**

named argument

Null

OLE object

OLE status code

placeholder

point

project

property

property page

RGB

run time

## **S-Z**

script

SendKeys statement

shortcut menu

system colors

tab order

tag

target

toolbox

transparent

UNC

URL

WWW

z-order

**browser**

Software that interprets the markup of HTML files posted on the World Wide Web, formats them into Web pages, and displays them. Some browsers can also open special programs to play sound or video files in Web documents if you have the necessary hardware. Internet Explorer is a browser.

## **tag**

Embedded between angle brackets in HTML text to add character or paragraph formatting to the text. Web browsers display text and graphic elements based on the tags an author uses. The tag itself is not displayed by the browser.

For example, the text

Make `<B>this text</B>` look bold

is displayed this way by a browser:

Make **this text** look bold

**mask**

Enables you to isolate parts of an image while you apply color changes or other effects to the rest of the image.

**modal**

A window or form state. A modal window must be closed before you can work in any other windows. Dialog boxes and messages are usually modal.

If a window is not modal, it can remain open while you work in other windows.

**home page**

The main page of a World Wide Web site. A home page is generated by a Web owner and often has links to other pages, both within and outside the site.



## **HTTP**

Hypertext Transfer Protocol. A World Wide Web standard for transferring data, such as text, graphics, sound, and other digital information, between Web servers and clients. URLs of files on Web servers begin with **http://**

**hyperlink**

A screen region that is sensitive to mouse clicks and that triggers a jump to related material in the same or another file. Hyperlinks are typically represented by colored and underlined text, or by a graphic, and they often change color after they have been used.

**hypertext**

A collection of documents containing cross-references or links which, with the aid of a browser, allow the reader to move easily within a document or from one document to another.

**Internet**

A worldwide network of thousands of smaller computer networks and millions of commercial, educational, government, and personal computers.

**intranet**

A network within an organization that uses Internet technologies (such as the HTTP or FTP protocols). You can use an intranet to navigate between documents, pages, or objects using hyperlinks.

**ActiveX Control Pad**

A Microsoft application used to create, open, modify, and save Web pages.

## **URL**

Uniform Resource Locator. Identifies the full path of a document, graphic, or other file on the Internet or on an intranet.

A URL expresses the protocol (such as FTP or HTTP) to be accessed and the file's location. A URL may also specify an Internet e-mail address or a newsgroup. Some examples of URLs are:

`http://www.someones.homepage/default.html`

`ftp://ftp.server.somewhere/ftp.file`

`gopher://server.name`

`file://Server/Share/File.doc`

**WWW**

World Wide Web. A system for navigating the Internet by using hyperlinks. When you use a Web browser, the Web appears as a collection of text, pictures, sounds, and digital movies.



**control**

A tool you select from the ActiveX Control Pad toolbox to draw an object, such as a **CommandButton** or a **TextBox**, in an HTML Layout.

Controls have their own set of recognized properties and events. You use controls to receive user input, display output, and trigger event procedures. You can manipulate most controls using methods.

**z-order**

The visual layering of controls on a form along the form's z-axis (depth). The z-order determines which controls are in front of other controls.

**toolbox**

A collection of controls you can select to draw an object, such as a **CommandButton** or a **TextBox**, in an HTML Layout.

## **method**

A statement in VBScript that performs an action for an object. The syntax for a method is:

*Object.Method [Value]*

For example, the following statement uses the **AddItem** method to place the word "Delete" in the List2 list box:

```
List2.AddItem "Delete"
```

**argument**

A constant, variable, or expression that supplies information to an action, procedure, or method.

For example, in the Beforeupdate event syntax,

**Private Sub *object*\_BeforeUpdate( *Cancel* As Boolean)**

*Cancel* is an argument

**event**

An occurrence, often initiated by the user, to which a program can respond. A key press, button push, and mouse movement are typical events.

**function**

A statement that returns a value to a script. The value returned can be assigned to a variable, a property, or another statement or function.

For example, in the **Picture** property syntax:

```
object.Picture = LoadPicture(pathname)
```

**LoadPicture** is a function that assigns an image to **Picture**.

**global variable**

A variable whose value can be accessed and modified by any event handler on a Web site.



**property**

An attribute of an object that you set to define one of the object's characteristics (such as size or color) or an aspect of its behavior (such as whether it is hidden).

**run time**

The time during which an application is running and you can interact with it as a user would. For example, during run time you can view an .alx file in a browser such as Internet Explorer.

In contrast, during design time you can create an application and modify its design.

**UNC**

Uniform Naming Convention. The UNC specifies a directory on a server on a local area network.

The basic format is:

`\\servername\sharename`

where "servername" is the host name of a network file server, and "sharename" is the name of a networked or shared directory.

**script**

In ActiveX Control Pad terminology, code written in VBScript or JavaScript. Inserted in an HTML page or HTML Layout, a script consists of a set of instructions that connect events with actions.

**GIF**

Graphics Interchange Format. A graphics file format that many World Wide Web browsers can display as inline graphics. GIF was developed specifically for transmitting images. It is best used for graphics with few colors, such as cartoons or line drawings. GIF files are compressed bitmaps. See JPEG

**HTML**

Hypertext Markup Language. A system of marking up, or tagging, a document so it can be published on the World Wide Web. Documents prepared in HTML include reference graphics and formatting tags. You use a Web browser (such as Microsoft Internet Explorer) to view these documents.

**JPEG**

Joint Photographic Experts Group. A graphics file format supported by many World Wide Web browsers. JPEG was developed for compressing and storing photographic images and is best used for graphics containing many colors, such as scanned photos. JPEG files, which have a .jpg extension in Windows, are compressed bitmaps. See GIF

**design time**

The time during which you can build or modify an application in the development environment by adding controls, setting control properties, and so on. For example, during design time you can edit an .alx file in HTML Layout.

In contrast, during run time you can interact with an application as a user would.



**COM object**

An object that conforms to the Component Object Model. COM defines how ActiveX objects and their clients interact within processes or across process boundaries.

**shortcut menu**

A list of commands that is displayed when you click the right mouse button. Shortcut menus provide quick access to frequently used commands that are also available from the main menu bar. The commands listed depend on the object you click.

**event handler**

Code that is executed when a particular event occurs. The following example shows the event handler for the **Click** event associated with **CommandButton1**. The event handler sets the **Caption** property of **BannerLabel**.

```
Sub CommandButton1_Click ()  
BannerLabel.Caption = "New Caption"  
End Sub
```

**default**

A predefined setting or value that is assumed if no other value is specified. For example, if you do not assign a value to the **Enabled** property of a **CheckBox**, the **Enabled** property is **True** by default.

## Bring to Front Action

```
{ewc HLP95EN.DLL, DYNALINK, "See Also:." isActBringToFrontActionC"} {ewc HLP95EN.DLL, DYNALINK, "Example:." isActBringToFrontActionX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To:." isActBringToFrontActionA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics:." isActBringToFrontActionS"}
```

The **Bring to Front** action positions a control within an HTML Layout on top of overlapping controls.

### List View format

When selecting the **Bring to Front** action, the name of the control you have selected appears under the Object category in the Script pane and the **Bring to Front** action appears under the Action category.

### Code View format

When selecting the **Bring to Front** action in Code View, an action appears in the Script pane with the following command:

#### VBScript:

```
call object.ZOrder(0)
```

#### JavaScript

```
object.ZOrder(0)
```

## Go To Page Action

```
{ewc HLP95EN.DLL, DYNALINK, "See Also:." isActGoToPageActionC"} {ewc HLP95EN.DLL, DYNALINK, "Example:." isActGoToPageActionX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To:." isActGoToPageActionA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics:." isActGoToPageActionS"}
```

The **Go To Page** action causes the browser to navigate to a specified web location. **Go To Page** appears as the first action in the Action pane, at the highest level, and is available when editing any file.

### List View format

When selecting the **Go To Page** action, the Text String Value dialog box appears. You can type in a URL or customized variable name and select **OK**. No object is shown under the Object category in the Script pane, and the **Go To Page** action appears under the Action category, along with the URL assignment in quotation marks.

### Code View format

When selecting the **Go To Page** action in Code View, an action appears in the Script pane in the following format with empty quotation marks:

```
Window.location.href = ""
```

You can type in a URL within the quotation marks.

## Hide Control Action

```
{ewc HLP95EN.DLL, DYNALINK, "See Also:" isActHideControlActionC"} {ewc HLP95EN.DLL, DYNALINK, "Example:" isActHideControlActionX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To:" isActHideControlActionA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics:" isActHideControlActionS"}
```

The **Hide Control** action makes a control invisible.

### List View format

When selecting the **Hide Control** action, the name of the control you have selected appears under the Object category in the Script pane and the **Hide Control** action appears under the Action category.

### Code View format

When selecting the **Hide Control** action in Code View, an action appears in the Script pane with the following Boolean assignment:

#### VBScript:

```
object.Visible = False
```

#### JavaScript

```
object.Visible = false
```

## Send to Back Action

```
{ewc HLP95EN.DLL, DYNALINK, "See Also:" isActSendtoBackActionC"} {ewc HLP95EN.DLL, DYNALINK, "Example:" isActSendtoBackActionX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To:" isActSendtoBackActionA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics:" isActSendtoBackActionS"}
```

The **Send to Back** action positions a control within an HTML Layout behind overlapping controls.

### List View format

When selecting the **Send to Back** action, the name of the control you have selected appears under the Object category in the Script pane and the **Send to Back** action appears under the Action category.

### Code View format

When using VBScript and selecting the **Send to Back** action in Code View, an action appears in the Script pane with the following command:

#### VBScript:

```
call object.ZOrder(1)
```

#### JavaScript

```
object.ZOrder(1)
```

## Show Control Action

```
{ewc HLP95EN.DLL, DYNALINK, "See Also:" isActShowControlActionC"} {ewc HLP95EN.DLL, DYNALINK, "Example:" isActShowControlActionX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To:" isActShowControlActionA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics:" isActShowControlActionS"}
```

The **Show Control** action makes a control visible.

### List View format

When selecting the **Show Control** action, the name of the control you have selected appears under the Object category in the Script pane and the **Show Control** action appears under the Action category.

### Code View format

When using VBScript and selecting the **Show Control** action in Code View, an action appears in the Script pane with the following Boolean assignment:

#### VBScript:

#### JavaScript

---

`object.Visible = True`

---

`object.Visible = true`

## CheckBox Control

```
{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3objCheckboxC "}          {ewc HLP95EN.DLL, DYNALINK,
"Example": "f3objCheckboxX ":1}          {ewc HLP95EN.DLL, DYNALINK, "Properties": "f3objCheckboxP "}          {ewc
HLP95EN.DLL, DYNALINK, "Methods": "f3objCheckboxM "}          {ewc HLP95EN.DLL, DYNALINK,
"Events": "f3objCheckboxE "}          {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3objCheckboxS "}
```

Displays the selection state of an item.

### Remarks

Use a **CheckBox** to give the user a choice between two values such as *Yes/No*, *True/False*, or *On/Off*. When the user selects a **CheckBox**, it displays a special mark (such as an X) and its current setting is *Yes*, *True*, or *On*; if the user does not select the **CheckBox**, it is empty and its setting is *No*, *False*, or *Off*. Depending on the value of the **TripleState** property, a **CheckBox** can also have a null value.

A disabled **CheckBox** shows the current value, but is dimmed and does not allow changes to the value from the user interface.

You can also group check boxes so that a user can select one or more of a group of related items. For example, you can create an order form that contains a list of available items, with a **CheckBox** preceding each item. The user can select a particular item or items by checking the corresponding **CheckBox**.

The default property of a **CheckBox** is the **Value** property.

The default event of a **CheckBox** is the Click event.

**Note** The **ListBox** also lets you put a check mark by selected options. Depending on your application, you can use the **ListBox** instead of using a group of **CheckBox** controls.



## ComboBox Control

```
{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3objComboBoxC "} {ewc HLP95EN.DLL, DYNALINK,
"Example":"f3objComboBoxX ":1} {ewc HLP95EN.DLL, DYNALINK, "Properties":"f3objComboBoxP "} {ewc
HLP95EN.DLL, DYNALINK, "Methods":"f3objComboBoxM "} {ewc HLP95EN.DLL, DYNALINK,
"Events":"f3objComboBoxE "} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3objComboBoxS "}
```

Combines the features of a **ListBox** and a **TextBox**. The user can enter a new value, as with a **TextBox**, or the user can select an existing value as with a **ListBox**.

### Remarks

The list in a **ComboBox** consists of rows of text. Each row can have one or more columns, which can appear with or without headings. Some applications do not support column headings, others provide only limited support.

The default property of a **ComboBox** is the **Value** property.

The default event of a **ComboBox** is the Change event.

**Note** If you want more than a single line of the list to appear at all times, you might want to use a **ListBox** instead of a **ComboBox**. If you want to use a **ComboBox** and limit values to those in the list, you can set the **Style** property of the **ComboBox** so the control looks like a drop-down list box.

## CommandButton Control

```
{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3objCommandButtonC "}      {ewc HLP95EN.DLL, DYNALINK,
"Example":"f3objCommandButtonX ":1}      {ewc HLP95EN.DLL, DYNALINK, "Properties":"f3objCommandButtonP "}
{ewc HLP95EN.DLL, DYNALINK, "Methods":"f3objCommandButtonM "}      {ewc HLP95EN.DLL, DYNALINK,
"Events":"f3objCommandButtonE "}      {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3objCommandButtonS "}
```

Starts, ends, or interrupts an action or series of actions.

### Remarks

The event procedure assigned to the **CommandButton's** Click event determines what the **CommandButton** does. For example, you can create a **CommandButton** that jumps to another HTML page. You can also display text, a picture, or both on a **CommandButton**.

The default property of a **CommandButton** is the **Value** property.

The default event for a **CommandButton** is the Click event.

## ListBox Control

```
{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3objListBoxC "}          {ewc HLP95EN.DLL, DYNALINK,  
"Example":"f3objListBoxX ":1}          {ewc HLP95EN.DLL, DYNALINK, "Properties":"f3objListBoxP "}          {ewc  
HLP95EN.DLL, DYNALINK, "Methods":"f3objListBoxM "}          {ewc HLP95EN.DLL, DYNALINK, "Events":"f3objListBoxE "} }  
{ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3objListBoxS " }
```

Displays a list of values and lets you select one or more entries from the list.

### Remarks

The **Listbox** can either appear as a list or as a group of **OptionButton** controls or **CheckBox** controls.

The default property for a **Listbox** is the **Value** property.

The default event for a **Listbox** is the Click event.

You can't drop text into a drop-down **Listbox**.

**Note** **Listbox** is a windowed control. Therefore you cannot position it behind a windowless control in the z-order. For example, you cannot position a **Listbox** behind a **CommandButton**. However, you can position the z-order of **Listbox** relative to other **Listbox** controls.

## OptionButton Control

```
{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3objOptionButtonC "}          {ewc HLP95EN.DLL, DYNALINK,
"Example": "f3objOptionButtonX ":1}          {ewc HLP95EN.DLL, DYNALINK, "Properties": "f3objOptionButtonP "}
{ewc HLP95EN.DLL, DYNALINK, "Methods": "f3objOptionButtonM "}          {ewc HLP95EN.DLL, DYNALINK,
"Events": "f3objOptionButtonE "}          {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3objOptionButtonS "}
```

Shows the selection status of one item in a group of choices.

### Remarks

Use an **OptionButton** to show whether a single item in a group is selected.

If the user selects the **OptionButton**, the current setting is *Yes*, *True*, or *On*; if the user does not select the **OptionButton**, the setting is *No*, *False*, or *Off*. For example, an **OptionButton** in an inventory-tracking application might show whether an item is discontinued. A disabled **OptionButton** is dimmed and does not show a value.

Depending on the value of the **TriState** property, an **OptionButton** can also have a null value.

You can also group **OptionButtons** so that a user can select one or more of a group of related items. For example, you can create an order form with a list of available items, with an **OptionButton** preceding each item. The user can select a particular item by checking the corresponding **OptionButton**.

The default property for an **OptionButton** is the **Value** property.

The default event for an **OptionButton** is the Click event.

## ScrollBar Control

```
{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3objScrollBarC "} {ewc HLP95EN.DLL, DYNALINK,
"Example": "f3objScrollBarX ": 1} {ewc HLP95EN.DLL, DYNALINK, "Properties": "f3objScrollBarP "} {ewc
HLP95EN.DLL, DYNALINK, "Methods": "f3objScrollBarM "} {ewc HLP95EN.DLL, DYNALINK, "Events": "f3objScrollBarE
"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3objScrollBarS "}
```

Returns or sets the value of another control based on the position of the scroll box.

### Remarks

A **ScrollBar** is a stand-alone control you can place on an HTML Layout. It is visually like the scroll bar you see in certain objects such as a **ListBox** or the drop-down portion of a **ComboBox**. However, unlike the scroll bars in these examples, the stand-alone **ScrollBar** is not an integral part of any other control.

To use the **ScrollBar** to set or read the value of another control, you must write code for the **ScrollBar's** events and methods. For example, to use the **ScrollBar** to update the value of a **TextBox**, you can write code that reads the **Value** property of the **ScrollBar** and then sets the **Value** property of the **TextBox**.

The default property for a **ScrollBar** is the **Value** property.

The default event for a **ScrollBar** is the Change event.

**Note** To create a horizontal or vertical **ScrollBar**, drag the sizing handles of the **ScrollBar** horizontally or vertically on the HTML Layout.

## SpinButton Control

```
{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3objSpinButtonC "}          {ewc HLP95EN.DLL, DYNALINK,
"Example":"f3objSpinButtonX ":1}          {ewc HLP95EN.DLL, DYNALINK, "Properties":"f3objSpinButtonP "}          {ewc
HLP95EN.DLL, DYNALINK, "Methods":"f3objSpinButtonM "}          {ewc HLP95EN.DLL, DYNALINK,
"Events":"f3objSpinButtonE "}          {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3objSpinButtonS "}
```

Increments and decrements numbers.

### Remarks

Clicking a **SpinButton** changes only the value of the **SpinButton**. You can write code that uses the **SpinButton** to update the displayed value of another control. For example, you can use a **SpinButton** to change the month, the day, or the year shown on a date. You can also use a **SpinButton** to scroll through a range of values or a list of items, or to change the value displayed in a text box.

To display a value updated by a **SpinButton**, you must assign the value of the **SpinButton** to the displayed portion of a control, such as the **Caption** property of a **Label** or the **Text** property of a **TextBox**. To create a horizontal or vertical **SpinButton**, drag the sizing handles of the **SpinButton** horizontally or vertically on the HTML Layout.

The default property for a **SpinButton** is the **Value** property.

The default event for a **SpinButton** is the Change event.

## TabStrip Control

```
{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3objTabStripC "} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3objTabStripX ":1} {ewc HLP95EN.DLL, DYNALINK, "Properties": "f3objTabStripP "} {ewc HLP95EN.DLL, DYNALINK, "Methods": "f3objTabStripM "} {ewc HLP95EN.DLL, DYNALINK, "Events": "f3objTabStripE "} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3objTabStripS "}
```

Presents a set of related controls as a visual group.

**TabStrip**

L

**Tabs (Tab)**

### Remarks

You can use a **TabStrip** to view different sets of information for related controls.

For example, the controls might represent information about a daily schedule for a group of individuals, with each set of information corresponding to a different individual in the group. Set the title of each tab to show one individual's name. Then, you can write code that, when you click a tab, updates the controls to show information about the person identified on the tab.

**Note** The **TabStrip** is implemented as a container of a **Tabs** collection, which in turn contains a group of **Tab** objects.

The default property for a **TabStrip** is the **SelectedItem** property.

The default event for a **TabStrip** is the Change event.

# TextBox Control

```
{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3objTextBoxC "} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3objTextBoxX ":1} {ewc HLP95EN.DLL, DYNALINK, "Properties": "f3objTextBoxP "} {ewc HLP95EN.DLL, DYNALINK, "Methods": "f3objTextBoxM "} {ewc HLP95EN.DLL, DYNALINK, "Events": "f3objTextBoxE "} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3objTextBoxS "}
```

Displays information from a user.

## Remarks

A **TextBox** is the control most commonly used to display information entered by a user.

Formatting applied to any piece of text in a **TextBox** will affect all text in the control. For example, if you change the font or point size of any character in the control, the change will affect all characters in the control.

The default property for a **TextBox** is the **Value** property.

The default event for a **TextBox** is the Change event.



## ToggleButton Control

```
{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3objToggleButtonC "}          {ewc HLP95EN.DLL, DYNALINK,
"Example": "f3objToggleButtonX ":1}          {ewc HLP95EN.DLL, DYNALINK, "Properties": "f3objToggleButtonP "}
{ewc HLP95EN.DLL, DYNALINK, "Methods": "f3objToggleButtonM "}          {ewc HLP95EN.DLL, DYNALINK,
"Events": "f3objToggleButtonE "}          {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3objToggleButtonS "}
```

Shows the selection state of an item.

### Remarks

Use a **ToggleButton** to show whether an item is selected. If the user selects the **ToggleButton**, the current setting is *Yes*, *True*, or *On*; if the user does not select the **ToggleButton**, the setting is *No*, *False*, or *Off*. A disabled **ToggleButton** shows a value, but is dimmed and does not allow changes from the user interface.

You can also group **ToggleButtons** so that a user can select one or more of a group of related items. For example, you can create an order form with a list of available items, with a **ToggleButton** preceding each item. The user can select a particular item by selecting the appropriate **ToggleButton**.

The default property of a **ToggleButton** is the **Value** property.

The default event of a **ToggleButton** is the Click event.

## Font Object

```
{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3objFontC "} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3objFontX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3objFontA"} {ewc HLP95EN.DLL, DYNALINK, "Properties":"f3objFontP "} {ewc HLP95EN.DLL, DYNALINK, "Methods":"f3objFontM "} {ewc HLP95EN.DLL, DYNALINK, "Events":"f3objFontE "} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3objFontS "}
```

Defines the characteristics of the text used by a control or HTML Layout.

**UserForm**

L

**Font**

### Remarks

Each control and HTML Layout has its own **Font** object to let you set its text characteristics independently of the characteristics defined for other controls and HTML Layouts. Use font properties to specify the font name, to set bold or underlined text, or to adjust the size of the text.

**Note** The font properties of your HTML Layout or container determine the default font attributes of controls you put on the HTML Layout.

The default property for the **Font** object is the **Name** property. If the **Name** property contains a null string, the **Font** object uses the default system font.

## Label Control

```
{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3objLabelC"}  
HLP95EN.DLL,DYNALINK,"Example":"f3objLabelX":1}  
{ewc HLP95EN.DLL,DYNALINK,"Properties":"f3objLabelP"}  
{ewc HLP95EN.DLL,DYNALINK,"Events":"f3objLabelE"}
```

```
{ewc  
{ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3objLabelA"}  
  {ewc HLP95EN.DLL,DYNALINK,"Methods":"f3objLabelM"}  
  {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3objLabelS"}
```

Displays descriptive text.

### Remarks

A **Label** control on an HTML Layout displays descriptive text such as titles, captions, pictures, or brief instructions. For example, labels for an address book might include a **Label** for a name, street, or city. A **Label** doesn't change as you move from record to record.

The default property for a **Label** is the **Caption** property.

The default event for a **Label** is the Click event.

## Tab Object

```
{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3objTabC "}          {ewc HLP95EN.DLL, DYNALINK, "Example": "f3objTabX  
":1}                {ewc HLP95EN.DLL, DYNALINK, "Properties": "f3objTabP "}          {ewc HLP95EN.DLL, DYNALINK,  
"Methods": "f3objTabM "}          {ewc HLP95EN.DLL, DYNALINK, "Events": "f3objTabE "}          {ewc HLP95EN.DLL,  
DYNALINK, "Specifics": "f3objTabS "}
```

A **Tab** is an individual member of a **Tabs** collection.

**TabStrip**

L

**Tabs {Tab}**

**Tab**

### Remarks

Visually, a **Tab** object appears as a rectangle protruding from a larger rectangular area or as a button adjacent to a rectangular area.

In contrast to an HTML Layout, a **Tab** does not contain any controls. Controls that appear within the region bounded by a **TabStrip** are contained on the HTML Layout, as is the **TabStrip**.

Each **Tab** has its own set of properties, but has no methods or events. You must use events from the appropriate **TabStrip** to initiate processing of an individual **Tab**.

Each **Tab** has a unique name and index value within the collection. You can reference a **Tab** by either its name or its index value. The index of the first **Tab** is 0; the index of the second **Tab** is 1; and so on. When two **Tab** objects have the same name, you must reference each **Tab** by its index value. References to the name in code will access only the first **Tab** that uses the name.

## Tabs Collection

```
{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3objTabCollC "} {ewc HLP95EN.DLL, DYNALINK,
"Example":"f3objTabCollX ":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3objTabCollA"} {ewc
HLP95EN.DLL, DYNALINK, "Properties":"f3objTabCollP "} {ewc HLP95EN.DLL, DYNALINK, "Methods":"f3objTabCollM
"} {ewc HLP95EN.DLL, DYNALINK, "Events":"f3objTabCollE "} {ewc HLP95EN.DLL, DYNALINK,
"Specifics":"f3objTabCollS "}
```

A **Tabs** collection includes all **Tabs** of a **TabStrip**.

### Remarks

Each **Tabs** collection provides the features to manage the number of tabs in the collection and to identify the tab that is currently in use.

The default value of the **Tabs** collection identifies the current **Tab** of a collection.

A **Tab** object has a unique name and index value within a **Tabs** collection. You can reference a **Tab** either by its name or its index value. The index value reflects the ordinal position of the **Tab** within the collection. The index of the first **Tab** in a collection is 0; the index of the second **Tab** is 1; and so on.

## AfterUpdate Event

```
{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3evtAfterUpdateC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3evtAfterUpdateX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3evtAfterUpdateA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3evtAfterUpdateS"}
```

Occurs after data in a control is changed through the user interface.

### Syntax

**Private Sub** *object*\_**AfterUpdate**( )

The **AfterUpdate** event syntax has these parts:

| Part          | Description               |
|---------------|---------------------------|
| <i>object</i> | Required. A valid object. |

### Remarks

This event cannot be canceled. If you want to cancel the update (to restore the previous value of the control), use the BeforeUpdate event and set the *Cancel* argument to **True**.

The AfterUpdate event occurs after the BeforeUpdate event and before the Exit event for the current control and before the Enter event for the next control in the tab order.

## BeforeDragOver Event

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3evtBeforeDragOverC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3evtBeforeDragOverX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3evtBeforeDragOverA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3evtBeforeDragOverS"}

Occurs when a drag-and-drop operation is in progress.

### Syntax

For TabStrip

```
Private Sub object_BeforeDragOver(index As Long, ByVal Cancel As MSForms.ReturnBoolean, ByVal Data As DataObject, ByVal X As Single, ByVal Y As Single, ByVal DragState As fmDragState, ByVal Effect As MSForms.ReturnEffect, ByVal Shift As fmShiftState)
```

For other controls

```
Private Sub object_BeforeDragOver(ByVal Cancel As MSForms.ReturnBoolean, ByVal Data As DataObject, ByVal X As Single, ByVal Y As Single, ByVal DragState As fmDragState, ByVal Effect As MSForms.ReturnEffect, ByVal Shift As fmShiftState)
```

The **BeforeDragOver** event syntax has these parts:

| Part             | Description  |
|------------------|--|
| <i>object</i>    | Required. A valid object name.   |
| <i>Cancel</i>    | Required. Event status. <b>False</b> indicates that the control should handle the event (default). <b>True</b> indicates the application handles the event.  |
| <i>Ctrl</i>      | Required. The control being dragged over.  |
| <i>Data</i>      | Required. Data that is dragged in a drag-and-drop operation. The data is packaged in a <b>DataObject</b> .   |
| <i>X, Y</i>      | Required. The horizontal and vertical coordinates of the control's position. Both coordinates are measured in points. <i>X</i> is measured from the left edge of the control; <i>Y</i> is measured from the top of the control.  |
| <i>DragState</i> | Required. Transition state of the data being dragged.  |
| <i>X, Y</i>      | Required. The horizontal and vertical coordinates of the control's position. Both coordinates are measured in points. <i>X</i> is measured from the left edge of the control; <i>Y</i> is measured from the top of the control.. |
| <i>Effect</i>    | Required. Operations supported by the <u>drop source</u> .   |
| <i>Shift</i>     | Required. Specifies the state of SHIFT, CTRL, and ALT.   |

### Settings

The settings for *DragState* are:

| Constant                | Value | Description  |
|-------------------------|-------|--|
| <i>fmDragStateEnter</i> | 0     | Mouse pointer is within range of a target.                                       |
| <i>FmDragStateLeave</i> | 1     | Mouse pointer is outside the range of a target.                                  |
| <i>FmDragStateOver</i>  | 2     | Mouse pointer is at a new position, but remains within range of the same target. |

The settings for *Effect* are:

| Constant                | Value | Description                    |
|-------------------------|-------|--------------------------------|
| <i>fmDropEffectNone</i> | 0     | Does not copy or move the drop |

|                               |   |   |
|-------------------------------|---|---|
|                               |   | source to the drop target.                          |
| <i>FmDropEffectCopy</i>       | 1 | Copies the drop source to the drop target.          |
| <i>FmDropEffectMove</i>       | 2 | Moves the drop source to the drop target.           |
| <i>FmDropEffectCopyOrMove</i> | 3 | Copies or moves the drop source to the drop target. |

The settings for *Shift* are:

| <b>Constant</b>    | <b>Value</b> | <b>Description</b> |
|--------------------|--------------|--------------------|
| <i>fmShiftMask</i> | 1            | SHIFT was pressed. |
| <i>FmCtrlMask</i>  | 2            | CTRL was pressed.  |
| <i>FmAltMask</i>   | 4            | ALT was pressed.   |

### Remarks

Use this event to monitor the mouse pointer as it enters, leaves, or rests directly over a valid target. When a drag-and-drop operation is in progress, the system initiates this event when the user moves the mouse, or presses or releases the mouse buttons. The mouse pointer position determines the target object that receives this event. You can determine the state of the mouse pointer by examining the *DragState* argument.

When a control handles this event, you can use the *Effect* argument to identify the drag-and-drop action to perform. When *Effect* is set to **fmDropEffectCopyOrMove**, the drop source supports a copy (**fmDropEffectCopy**), move (**fmDropEffectMove**), or a cancel (**fmDropEffectNone**) operation.

When *Effect* is set to **fmDropEffectCopy**, the drop source supports a copy or a cancel (**fmDropEffectNone**) operation.

When *Effect* is set to **fmDropEffectMove**, the drop source supports a move or a cancel (**fmDropEffectNone**) operation.

When *Effect* is set to **fmDropEffectNone**, the drop source supports a cancel operation.

Most controls do not support drag-and-drop while *Cancel* is **False**, which is the default setting. This means the control rejects attempts to drag or drop anything on the control, and the control does not initiate the *BeforeDropOrPaste* event. The **TextBox** and **ComboBox** controls are exceptions to this; these controls support drag-and-drop operations even when *Cancel* is **False**.



# BeforeDropOrPaste Event

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3evtBeforeDropOrPasteC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3evtBeforeDropOrPasteX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3evtBeforeDropOrPasteA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3evtBeforeDropOrPasteS"}

Occurs when the user is about to drop or paste data onto an object.

## Syntax

For TabStrip

```
Private Sub object_BeforeDropOrPaste(index As Long, ByVal Cancel As MSForms.ReturnBoolean, ByVal Action As fmAction, ByVal Data As DataObject, ByVal X As Single, ByVal Y As Single, ByVal Effect As MSForms.ReturnEffect, ByVal Shift As fmShiftState)
```

For other controls

```
Private Sub object_BeforeDropOrPaste(ByVal Cancel As MSForms.ReturnBoolean, ByVal Action As fmAction, ByVal Data As DataObject, ByVal X As Single, ByVal Y As Single, ByVal Effect As MSForms.ReturnEffect, ByVal Shift As fmShiftState)
```

The **BeforeDropOrPaste** event syntax has these parts:

| Part          | Description   |
|---------------|---|
| <i>object</i> | Required. A valid object name.  |
| <i>index</i>  | Required. The index of the control that the drop or paste operation will affect.  |
| <i>Cancel</i> | Required. Event status. <b>False</b> indicates that the control should handle the event (default). <b>True</b> indicates the application handles the event.   |
| <i>ctrl</i>   | Required. The target control.   |
| <i>Action</i> | Required. Indicates the result, based on the current keyboard settings, of the pending drag-and-drop operation.   |
| <i>Data</i>   | Required. Data that is dragged in a drag-and-drop operation. The data is packaged in a <b>DataObject</b> .  |
| <i>X, Y</i>   | Required. The horizontal and vertical position of the mouse pointer when the drop occurs. Both coordinates are measured in points. <i>X</i> is measured from the left edge of the control; <i>Y</i> is measured from the top of the control.. |
| <i>Effect</i> | Required. Effect of the drag-and-drop operation on the target control.  |
| <i>Shift</i>  | Required. Specifies the state of SHIFT, CTRL, and ALT.  |

## Settings

The settings for *Action* are:

| Constant                | Value | Description   |
|-------------------------|-------|---|
| <i>fmActionPaste</i>    | 2     | Pastes the selected object into the drop target.  |
| <i>fmActionDragDrop</i> | 3     | Indicates the user has dragged the object from its source to the drop target and dropped it on the drop target. |

The settings for *Effect* are:

| Constant                | Value | Description                           |
|-------------------------|-------|---------------------------------------|
| <i>fmDropEffectNone</i> | 0     | Does not copy or move the <u>drop</u> |

|                               |   |   |
|-------------------------------|---|---|
|                               |   | <u>source</u> to the drop target.                   |
| <i>fmDropEffectCopy</i>       | 1 | Copies the drop source to the drop target.          |
| <i>fmDropEffectMove</i>       | 2 | Moves the drop source to the drop target.           |
| <i>fmDropEffectCopyOrMove</i> | 3 | Copies or moves the drop source to the drop target. |

The settings for *Shift* are:

| <b>Constant</b>    | <b>Value</b> | <b>Description</b> |
|--------------------|--------------|--------------------|
| <i>fmShiftMask</i> | 1            | SHIFT was pressed. |
| <i>fmCtrlMask</i>  | 2            | CTRL was pressed.  |
| <i>fmAltMask</i>   | 4            | ALT was pressed.   |

### Remarks

For a **TabStrip**, VBScript initiates this event when it transfers a data object to the control.

For other controls, the system initiates this event immediately prior to the drop or paste operation.

When a control handles this event, you can update the *Action* argument to identify the drag-and-drop action to perform. When *Effect* is set to **fmDropEffectCopyOrMove**, you can assign *Action* to **fmDropEffectNone**, **fmDropEffectCopy**, or **fmDropEffectMove**. When *Effect* is set to **fmDropEffectCopy** or **fmDropEffectMove**, you can reassign *Action* to **fmDropEffectNone**. You cannot reassign *Action* when *Effect* is set to **fmDropEffectNone**.

## BeforeUpdate Event

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3evtBeforeUpdateC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3evtBeforeUpdateX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3evtBeforeUpdateA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3evtBeforeUpdateS"}

Occurs before data in a control is changed.

### Syntax

**Private Sub *object*\_BeforeUpdate( *Cancel* As MSForms.ReturnBoolean)**

The **BeforeUpdate** event syntax has these parts:

| Part          | Description  |
|---------------|--|
| <i>object</i> | Required. A valid object.  |
| <i>Cancel</i> | Required. Event status. <b>False</b> indicates that the control should handle the event (default). <b>True</b> cancels the update and indicates the application should handle the event. |

### Remarks

This event occurs before the AfterUpdate and Exit events for the control (and before the Enter event for the next control that receives the focus).

If you set the *Cancel* argument to **True**, the focus remains on the control and neither the AfterUpdate event nor the Exit event occurs.

# Change Event

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3evtChangeC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3evtChangeX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3evtChangeA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3evtChangeS"}

Occurs when the **Value** property changes.

## Syntax

**Private Sub** *object\_Change*( )

The **Change** event syntax has these parts:

| Part          | Description               |
|---------------|---------------------------|
| <i>object</i> | Required. A valid object. |

## Settings

The Change event occurs when the setting of the **Value** property changes, regardless of whether the change results from execution of code or a user action in the interface.

Here are some examples of actions that change the **Value** property:

- Clicking a **CheckBox**, **OptionButton**, or **ToggleButton**.
- Entering or selecting a new text value for a **ComboBox**, **ListBox**, or **TextBox**.
- Selecting a different tab on a **TabStrip**.
- Moving the scroll box in a **ScrollBar**.
- Clicking the Up Arrow or Down Arrow on a **SpinButton**.

## Remarks

The Change event procedure can synchronize or coordinate data displayed among controls. For example, you can use the Change event procedure of a **ScrollBar** to update the contents of a **TextBox** that displays the value of the **ScrollBar**. Or you can use a Change event procedure to display data and formulas in a work area and results in another area.

**Note** In some cases, the Click event may also occur when the **Value** property changes. However, using the Change event is the preferred technique for detecting a new value for a property.

# Click Event

```
{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3evtClickC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3evtClickX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3evtClickA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3evtClickS"}
```

Occurs in one of two cases:

- The user definitively selects a value for a control with more than one possible value.
- The user clicks a control with the mouse.

## Syntax

For all controls

```
Private Sub object_Click( )
```

The **Click** event syntax has these parts:

| Part          | Description               |
|---------------|---------------------------|
| <i>object</i> | Required. A valid object. |

## Remarks

Of the two cases where the Click event occurs, the first case applies to the **CommandButton**, **Image**, **Label**, **ScrollBar**, and **SpinButton**. The second case applies to the **CheckBox**, **ComboBox**, **ListBox**, **TabStrip**, **TextBox**, and **ToggleButton**.

The following are examples of actions that initiate the Click event:

- Clicking a blank area of an HTML Layout or a disabled control (other than a list box) on the HTML Layout.
- Clicking a **CommandButton**. If the command button doesn't already have the focus, the Enter event occurs before the Click event.
- Pressing the SPACEBAR when a **CommandButton** has the focus.
- Clicking a control with the left mouse button (left-clicking).
- Pressing a control's accelerator key.

When the Click event results from clicking a control, the sequence of events leading to the Click event is:

1. MouseDown
2. MouseUp
3. Click

For some controls, the Click event occurs when the **Value** property changes. However, using the Change event is the preferred technique for detecting a new value for a property. The following are examples of actions that initiate the Click event due to assigning a new value to a control:

- Clicking a **CheckBox** or **ToggleButton**, pressing the SPACEBAR when one of these controls has the focus, pressing the accelerator key for one of these controls, or changing the value of the control in code.
- Changing the value of an **OptionButton** to **True**. Setting one **OptionButton** in a group to **True** sets all other buttons in the group to **False**, but the Click event occurs only for the button whose value changes to **True**.
- Selecting a value for a **ComboBox** or **ListBox** so that it unquestionably matches an item in the control's drop-down list. For example, if a list is not sorted, the first match for characters typed in the edit region may not be the only match in the list, so choosing such a value does not initiate the Click event. In a sorted list, you can use entry-matching to ensure that a selected value is a unique match for text the user types.

The Click event is not initiated when **Value** is set to **Null**.

**Note** Left-clicking changes the value of a control, thus it initiates the Click event. Right-clicking does not change the value of the control, so it does not initiate the Click event.

## DbIClick Event

```
{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3evtDbIClickC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3evtDbIClickX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3evtDbIClickA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3evtDbIClickS"}
```

Occurs when the user points to an object and then clicks a mouse button twice.

### Syntax

For TabStrip

```
Private Sub object_DbIClick(index As Long, Cancel As MSForms.ReturnBoolean)
```

For other controls

```
Private Sub object_DbIClick(Cancel As MSForms.ReturnBoolean)
```

The **DbIClick** event syntax has these parts:

| Part          | Description   |
|---------------|---|
| <i>object</i> | Required. A valid object.   |
| <i>index</i>  | Required. The position of a <b>Tab</b> object within a <b>Tabs</b> collection.  |
| <i>Cancel</i> | Required. Event status. <b>False</b> indicates that the control should handle the event (default). <b>True</b> indicates the application handles the event. |

### Remarks

For this event to occur, the two clicks must occur within the time span specified by the system's double-click speed setting.

For controls that support Click, the following sequence of events leads to the DbIClick event:

1. MouseDown
2. MouseUp
3. Click
4. DbIClick

If a control, such as **TextBox**, does not support Click, Click is omitted from the order of events leading to the DbIClick event.

If the return value of *Cancel* is **True** when the user clicks twice, the control ignores the second click. This is useful if the second click reverses the effect of the first, such as double-clicking a toggle button. The *Cancel* argument allows your HTML Layout to ignore the second click, so that either clicking or double-clicking the button has the same effect.

## DropButtonClick Event

```
{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3evtDropButtonClickC"} {ewc HLP95EN.DLL, DYNALINK,  
"Example": "f3evtDropButtonClickX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3evtDropButtonClickA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3evtDropButtonClickS"}
```

Occurs whenever the drop-down list appears or disappears.

### Syntax

**Private Sub** *object*\_**DropButtonClick**( )

The **DropButtonClick** event syntax has these parts:

| Part          | Description               |
|---------------|---------------------------|
| <i>object</i> | Required. A valid object. |

### Remarks

You can initiate the DropButtonClick event through code or by taking certain actions in the user interface.

In code, calling the **DropDown** method initiates the DropButtonClick event.

In the user interface, any of the following actions initiates the event:

- Clicking the drop-down button on the control.
- Pressing F4.

Any of the above actions, in code or in the interface, cause the drop-down box to appear on the control. The system initiates the DropButtonClick event when the drop-down box goes away.



## Enter, Exit Events

```
{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3evtEnterC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3evtEnterX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3evtEnterA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3evtEnterS"}
```

Enter occurs before a control actually receives the focus from a control on the same HTML Layout.  
Exit occurs immediately before a control loses the focus to another control on the same HTML Layout.

### Syntax

**Private Sub** *object*\_Enter( )

**Private Sub** *object*\_Exit( *Cancel* As MSForms.ReturnBoolean)

The **Enter** and **Exit** event syntaxes have these parts:

| Part          | Description  |
|---------------|--|
| <i>object</i> | Required. A valid object name.   |
| <i>Cancel</i> | Required. Event status. <b>False</b> indicates that the control should handle the event (default). <b>True</b> indicates the application handles the event and the focus should remain at the current control. |

### Remarks

The Enter and Exit events are similar to the GotFocus and LostFocus events in VBScript. Unlike GotFocus and LostFocus, the Enter and Exit events don't occur when an HTML Layout receives or loses the focus.

For example, suppose you select the check box that initiates the Enter event. If you then select another control in the same HTML Layout, the Exit event is initiated for the check box (because the focus is moving to a different object in the same HTML Layout) and then the Enter event occurs for the second control on the HTML Layout.

Because the Enter event occurs before the focus moves to a particular control, you can use an Enter event procedure to display instructions; for example, you could use an event procedure to display a small HTML Layout or message box identifying the type of data the control typically contains.

**Note** To prevent the control from losing focus, assign **True** to the *Cancel* argument of the Exit event.

## Error Event

```
{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3evtErrorC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3evtErrorX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3evtErrorA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3evtErrorS"}
```

Occurs when a control detects an error and cannot return the error information to a calling program.

### Syntax

```
Private Sub object_Error( ByVal Number As Integer, Description As MSForms.ReturnString,  
ByVal SCode As SCode, ByVal Source As String, ByVal HelpFile As String, ByVal HelpContext  
As Long, CancelDisplay As MSForms.ReturnBoolean)
```

The **Error** event syntax has these parts:

| <b>Part</b>          | <b>Description</b>  |
|----------------------|---|
| <i>object</i>        | Required. A valid object name.  |
| <i>Number</i>        | Required. Specifies a unique value that the control uses to identify the error.   |
| <i>Description</i>   | Required. A textual description of the error.   |
| <i>SCode</i>         | Required. Specifies the <a href="#">OLE status code</a> for the error. The low-order 16 bits specify a value that is identical to the <i>Number</i> argument. |
| <i>Source</i>        | Required. The string that identifies the control which initiated the event.   |
| <i>HelpFile</i>      | Required. Specifies a fully qualified path name for the Help file that describes the error.   |
| <i>HelpContext</i>   | Required. Specifies the <a href="#">context ID</a> of the Help file topic that contains a description of the error.   |
| <i>CancelDisplay</i> | Required. Specifies whether to display the error string in a message box.   |

### Remarks

The code written for the Error event determines how the control responds to the error condition.

The ability to handle error conditions varies from one application to another. The Error event is initiated when an error occurs that the application is not equipped to handle.

## KeyDown, KeyUp Events

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3evtKeyDownC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3evtKeyDownX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3evtKeyDownA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3evtKeyDownS"}

Occur in sequence when a user presses and releases a key. KeyDown occurs when the user presses a key. KeyUp occurs when the user releases a key.

### Syntax

**Private Sub** *object*\_**KeyDown**( *KeyCode* **As** MSForms.ReturnInteger, **ByVal** *Shift* **As** fmShiftState)

**Private Sub** *object*\_**KeyUp**( *KeyCode* **As** Integer, **ByVal** *Shift* **As** fmShiftState)

The **KeyDown** and **KeyUp** event syntaxes have these parts:

| Part           | Description  |
|----------------|--|
| <i>object</i>  | Required. A valid object name.   |
| <i>KeyCode</i> | Required. An integer that represents the key code of the key that was pressed or released. |
| <i>Shift</i>   | Required. The state of SHIFT, CTRL, and ALT.   |

### Settings

The settings for *Shift* are:

| Constant           | Value | Description        |
|--------------------|-------|--------------------|
| <i>fmShiftMask</i> | 1     | SHIFT was pressed. |
| <i>fmCtrlMask</i>  | 2     | CTRL was pressed.  |
| <i>fmAltMask</i>   | 4     | ALT was pressed.   |

### Remarks

The KeyDown event occurs when the user presses a key on a running HTML Layout while that HTML Layout or a control on it has the focus. The KeyDown and KeyPress events alternate repeatedly until the user releases the key, at which time the KeyUp event occurs. The HTML Layout or control with the focus receives all keystrokes. An HTML Layout can have the focus only if it has no controls or all its visible controls are disabled.

These events also occur if you send a keystroke to an HTML Layout or control using the SendKeys statement in VBScript.

The KeyDown and KeyUp events are typically used to recognize or distinguish between:

- Extended character keys, such as function keys.
- Navigation keys, such as HOME, END, PAGEUP, PAGEDOWN, UP ARROW, DOWN ARROW, RIGHT ARROW, LEFT ARROW, and TAB.
- Combinations of keys and standard keyboard modifiers (SHIFT, CTRL, or ALT).
- The numeric keypad and keyboard number keys.

The KeyDown and KeyUp events do not occur under the following circumstances:

The KeyDown and KeyPress events occur when you press or send an ANSI key. The KeyUp event occurs after any event for a control caused by pressing or sending the key. If a keystroke causes the focus to move from one control to another control, the KeyDown event occurs for the first control, while the KeyPress and KeyUp events occur for the second control.

The sequence of keyboard-related events is:

1. KeyDown
2. KeyPress
3. KeyUp

**Note** The KeyDown and KeyUp events apply only to HTML Layouts and controls on an HTML Layout. To interpret ANSI characters or to find out the ANSI character corresponding to the key pressed, use the KeyPress event.

# KeyPress Event

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3evtKeyPressC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3evtKeyPressX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3evtKeyPressA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3evtKeyPressS"}

Occurs when the user presses an ANSI key.

## Syntax

**Private Sub** *object\_KeyPress*( *KeyANSI* As MSForms.ReturnInteger)

The **KeyPress** event syntax has these parts:

| Part           | Description  |
|----------------|--|
| <i>object</i>  | Required. A valid object.  |
| <i>KeyANSI</i> | Required. An integer value that represents a standard numeric ANSI key code. |

## Remarks

The KeyPress event occurs when the user presses a key that produces a typeable character (an ANSI key) on a running HTML Layout while the HTML Layout or a control on it has the focus. The event can occur either before or after the key is released. This event also occurs if you send an ANSI keystroke to an HTML Layout or control using the SendKeys statement in VBScript.

A KeyPress event can occur when any of the following keys are pressed:

- Any printable keyboard character.
- CTRL combined with a character from the standard alphabet.
- CTRL combined with any special character.
- BACKSPACE.
- ESC.

A KeyPress event does not occur under the following conditions:

- Pressing TAB.
- Pressing ENTER.
- Pressing an arrow key.
- When a keystroke causes the focus to move from one control to another.

**Note** BACKSPACE is part of the ANSI Character Set, but DELETE is not. Deleting a character in a control using BACKSPACE causes a KeyPress event; deleting a character using DELETE doesn't.

When a user holds down a key that produces an ANSI keycode, the KeyDown and KeyPress events alternate repeatedly. When the user releases the key, the KeyUp event occurs. The HTML Layout or control with the focus receives all keystrokes. An HTML Layout can have the focus only if it has no controls, or if all its visible controls are disabled.

The default action for the KeyPress event is to process the event code that corresponds to the key that was pressed. *KeyANSI* indicates the ANSI character that corresponds to the pressed key or key combination. The KeyPress event interprets the uppercase and lowercase of each character as separate key codes and, therefore, as two separate characters.

To respond to the physical state of the keyboard, or to handle keystrokes not recognized by the KeyPress event, such as function keys, navigation keys, and any combinations of these with keyboard modifiers (ALT, SHIFT, or CTRL), use the KeyDown and KeyUp event procedures.

The sequence of keyboard-related events is:

1. KeyDown

2. KeyPress
3. KeyUp

# MouseDown, MouseUp Events

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3evtMouseDownC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3evtMouseDownX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3evtMouseDownA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3evtMouseDownS"}

Occur when the user clicks a mouse button. MouseDown occurs when the user presses the mouse button; MouseUp occurs when the user releases the mouse button.

## Syntax

For TabStrip

```
Private Sub object_MouseDown(index As Long, ByVal Button As fmButton, ByVal Shift As fmShiftState, ByVal X As Single, ByVal Y As Single)
```

```
Private Sub object_MouseUp(index As Long, ByVal Button As fmButton, ByVal Shift As fmShiftState, ByVal X As Single, ByVal Y As Single)
```

For other controls

```
Private Sub object_MouseDown(ByVal Button As fmButton, ByVal Shift As fmShiftState, ByVal X As Single, ByVal Y As Single)
```

```
Private Sub object_MouseUp(ByVal Button As fmButton, ByVal Shift As fmShiftState, ByVal X As Single, ByVal Y As Single)
```

The **MouseDown** and **MouseUp** event syntaxes have these parts:

| Part          | Description   |
|---------------|---|
| <i>object</i> | Required. A valid object.   |
| <i>index</i>  | Required. The index of the tab in a <b>TabStrip</b> with the specified event.                           |
| <i>Button</i> | Required. An integer value that identifies which mouse button caused the event.                         |
| <i>Shift</i>  | Required. The state of SHIFT, CTRL, and ALT.  |
| <i>X, Y</i>   | Required. The horizontal or vertical position, in points, from the left or top edge of the HTML Layout. |

## Settings

The settings for *Button* are:

| Constant              | Value | Description                    |
|-----------------------|-------|--------------------------------|
| <i>fmButtonLeft</i>   | 1     | The left button was pressed.   |
| <i>fmButtonRight</i>  | 2     | The right button was pressed.  |
| <i>fmButtonMiddle</i> | 4     | The middle button was pressed. |

The settings for *Shift* are:

| Value | Description                        |
|-------|------------------------------------|
| 1     | SHIFT was pressed.                 |
| 2     | CTRL was pressed.                  |
| 3     | SHIFT and CTRL were pressed.       |
| 4     | ALT was pressed.                   |
| 5     | ALT and SHIFT were pressed.        |
| 6     | ALT and CTRL were pressed.         |
| 7     | ALT, SHIFT, and CTRL were pressed. |

You can identify individual keyboard modifiers by using the following constants:

| Constant           | Value | Description           |
|--------------------|-------|-----------------------|
| <i>fmShiftMask</i> | 1     | Mask to detect SHIFT. |
| <i>fmCtrlMask</i>  | 2     | Mask to detect CTRL.  |
| <i>fmAltMask</i>   | 4     | Mask to detect ALT.   |

### Remarks

For a **TabStrip**, the index argument identifies the tab where the user clicked. An index of  $-1$  indicates the user did not click a tab. For example, if there are no tabs in the upper-right corner of the control, clicking in the upper-right corner sets the index to  $-1$ .

For an HTML Layout, the user can generate MouseDown and MouseUp events by pressing and releasing a mouse button in a blank area, record selector, or scroll bar on the HTML Layout.

The sequence of mouse-related events is:

1. MouseDown
2. MouseUp
3. Click
4. DbClick
5. MouseUp

MouseDown or MouseUp event procedures specify actions that occur when a mouse button is pressed or released. MouseDown and MouseUp events enable you to distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

If a mouse button is pressed while the pointer is over an HTML Layout or control, that object "captures" the mouse and receives all mouse events up to and including the last MouseUp event. This implies that the X, Y mouse-pointer coordinates returned by a mouse event may not always be within the boundaries of the object that receives them.

If mouse buttons are pressed in succession, the object that captures the mouse receives all successive mouse events until all buttons are released.

Use the *Shift* argument to identify the state of SHIFT, CTRL, and ALT when the MouseDown or MouseUp event occurred. For example, if both CTRL and ALT are pressed, the value of *Shift* is 6.



# MouseMove Event

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3evtMouseMoveC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3evtMouseMoveX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3evtMouseMoveA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3evtMouseMoveS"}

Occurs when the user moves the mouse.

## Syntax

For TabStrip

```
Private Sub object_MouseMove(index As Long, ByVal Button As fmButton, ByVal Shift As fmShiftState, ByVal X As Single, ByVal Y As Single)
```

For other controls

```
Private Sub object_MouseMove(ByVal Button As fmButton, ByVal Shift As fmShiftState, ByVal X As Single, ByVal Y As Single)
```

The **MouseMove** event syntax has these parts:

| Part          | Description  |
|---------------|--|
| <i>object</i> | Required. A valid object name.   |
| <i>index</i>  | Required. The index of the tab in a <b>TabStrip</b> associated with this event.                              |
| <i>Button</i> | Required. An integer value that identifies the state of the mouse buttons.                                   |
| <i>Shift</i>  | Required. Specifies the state of SHIFT, CTRL, and ALT.   |
| <i>X, Y</i>   | Required. The horizontal or vertical position, measured in points, from the left or top edge of the control. |

## Settings

The *index* argument specifies which tab was clicked over. A -1 designates that the user did not click on any of the tabs.

The settings for *Button* are:

| Value | Description                               |
|-------|---|
| 0     | No button is pressed.                     |
| 1     | The left button is pressed.               |
| 2     | The right button is pressed.              |
| 3     | The right and left buttons are pressed.   |
| 4     | The middle button is pressed.             |
| 5     | The middle and left buttons are pressed.  |
| 6     | The middle and right buttons are pressed. |
| 7     | All three buttons are pressed.            |

The settings for *Shift* are:

| Value | Description                  |
|-------|------------------------------|
| 1     | SHIFT was pressed.           |
| 2     | CTRL was pressed.            |
| 3     | SHIFT and CTRL were pressed. |
| 4     | ALT was pressed.             |
| 5     | ALT and SHIFT were pressed.  |
| 6     | ALT and CTRL were pressed.   |

7 ALT, SHIFT, and CTRL were pressed.

You can identify individual keyboard modifiers by using the following constants:

| <b>Constant</b>    | <b>Value</b> | <b>Description</b>    |
|--------------------|--------------|-----------------------|
| <i>fmShiftMask</i> | 1            | Mask to detect SHIFT. |
| <i>fmCtrlMask</i>  | 2            | Mask to detect CTRL.  |
| <i>fmAltMask</i>   | 4            | Mask to detect ALT.   |

### **Remarks**

The `MouseMove` event applies to HTML Layouts, controls on an HTML Layout, and labels.

`MouseMove` events are generated continually as the mouse pointer moves across objects. Unless another object has captured the mouse, an object recognizes a `MouseMove` event whenever the mouse position is within its borders.

Moving an HTML Layout can also generate a `MouseMove` event even if the mouse is stationary. `MouseMove` events are generated when the HTML Layout moves underneath the pointer. If an event procedure moves an HTML Layout in response to a `MouseMove` event, the event can continually generate (cascade) `MouseMove` events.

If two controls are very close together, and you move the mouse pointer quickly over the space between them, the `MouseMove` event might not occur for that space. In such cases, you might need to respond to the `MouseMove` event in both controls.

You can use the value returned in the *Button* argument to identify the state of the mouse buttons.

Use the *Shift* argument to identify the state of SHIFT, CTRL, and ALT when the `MouseMove` event occurred. For example, if both CTRL and ALT are pressed, the value of *Shift* is 6.

**Note** You can use `MouseDown` and `MouseUp` event procedures to respond to events caused by pressing and releasing mouse buttons.

# Scroll Event

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3evtScrollC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3evtScrollX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3evtScrollA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3evtScrollS"}

Occurs when the scroll box is repositioned.

## Syntax

For ScrollBar

```
Private Sub object_Scroll( )
```

For MultiPage

```
Private Sub object_Scroll( index As Long, ActionX As fmScrollAction, ActionY As fmScrollAction, ByVal RequestDx As Single, ByVal RequestDy As Single, ActualDx As MSForms.ReturnSingle, ActualDy As MSForms.ReturnSingle)
```

For Frame

```
Private Sub object_Scroll( ActionX As fmScrollAction, ActionY As fmScrollAction, ByVal RequestDx As Single, ByVal RequestDy As Single, ActualDx As MSForms.ReturnSingle, ActualDy As MSForms.ReturnSingle)
```

The **Scroll** event syntax has these parts:

| Part             | Description   |
|------------------|---|
| <i>object</i>    | Required. A valid object name.  |
| <i>index</i>     | Required. The index of the page in a <b>MultiPage</b> associated with this event.               |
| <i>ActionX</i>   | Required. The action that occurred in the horizontal direction.                                 |
| <i>ActionY</i>   | Required. The action that occurred in the vertical direction.                                   |
| <i>RequestDx</i> | Required. The distance, in points, you want the scroll bar to move in the horizontal direction. |
| <i>RequestDy</i> | Required. The distance, in points, you want the scroll bar to move in the vertical direction.   |
| <i>ActualDx</i>  | Required. The distance, in points, the scroll bar travelled in the horizontal direction.        |
| <i>ActualDy</i>  | Required. The distance, in points, the scroll bar travelled in the vertical direction.          |

## Settings

The settings for *ActionX* and *ActionY* are:

| Constant                      | Value | Description   |
|-------------------------------|-------|---|
| <i>fmScrollActionNoChange</i> | 0     | No change occurred.   |
| <i>fmScrollActionLineUp</i>   | 1     | A small distance up on a vertical scroll bar; a small distance to the left on a horizontal scroll bar. Movement is equivalent to pressing the up or left arrow keys on the keyboard to move the scroll bar. |
| <i>fmScrollActionLineDown</i> | 2     | A small distance down on a vertical scroll bar; a small distance to the right on a horizontal scroll bar. Movement is equivalent to pressing the  |

|                                     |    |  |
|-------------------------------------|----|--|
|                                     |    | down or right arrow keys on the keyboard to move the scroll bar.   |
| <i>fmScrollActionPageUp</i>         | 3  | One page up on a vertical scroll bar; one page to the left on a horizontal scroll bar. Movement is equivalent to pressing PAGE UP on the keyboard to move the scroll bar.  |
| <i>fmScrollActionPageDown</i>       | 4  | One page down on a vertical scroll bar; one page to the right on a horizontal scroll bar. Movement is equivalent to pressing PAGE DOWN on the keyboard to move the scroll bar.                                       |
| <i>fmScrollActionBegin</i>          | 5  | The top of a vertical scroll bar; the left end of a horizontal scroll bar.   |
| <i>fmScrollActionEnd</i>            | 6  | The bottom of a vertical scroll bar; the right end of a horizontal scroll bar.   |
| <i>fmScrollActionPropertyChange</i> | 8  | The value of either the <b>ScrollTop</b> or the <b>ScrollLeft</b> property changed. The direction and amount of movement depend on which property was changed and on the new property value.                         |
| <i>fmScrollActionControlRequest</i> | 9  | A control asked its container to scroll. The amount of movement depends on the specific control and container involved.  |
| <i>fmScrollActionFocusRequest</i>   | 10 | The user moved to a different control. The amount of movement depends on the placement of the selected control, and generally has the effect of moving the selected control so it is completely visible to the user. |

### Remarks

The Scroll events associated with a form, **Frame**, or **Page** return the following arguments: *ActionX*, *ActionY*, *ActualX*, and *ActualY*. *ActionX* and *ActionY* identify the action that occurred. *ActualX* and *ActualY* identify the distance that the scroll box traveled.

The default action is to calculate the new position of the scroll box and then scroll to that position.

You can initiate a Scroll event by issuing a **Scroll** method for a form, **Frame**, or **Page**. Users can generate Scroll events by moving the scroll box.

The Scroll event associated with the stand-alone **ScrollBar** indicates that the user moved the scroll box in either direction. This event is not initiated when the value of the **ScrollBar** changes by code or by the user clicking on parts of the **ScrollBar** other than the scroll box.

## SpinDown, SpinUp Events

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3evtSpinDownC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3evtSpinDownX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3evtSpinDownA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3evtSpinDownS"}

SpinDown occurs when the user clicks the lower or left spin-button arrow. SpinUp occurs when the user clicks the upper or right spin-button arrow.

### Syntax

**Private Sub** *object*\_SpinDown( )

**Private Sub** *object*\_SpinUp( )

The **SpinDown** and **SpinUp** event syntaxes have these parts:

| <b>Part</b>   | <b>Description</b>        |
|---------------|---------------------------|
| <i>object</i> | Required. A valid object. |

### Remarks

The SpinDown event decreases the **Value** property. The SpinUp event increases **Value**.

## Add Method

```
{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3mthAddC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3mthAddX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3mthAddA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3mthAddS"}
```

Adds or inserts a **Tab** in a **TabStrip**, or adds a control by its programmatic identifier (*ProgID*) to an HTML Layout.

### Syntax

For TabStrip

```
Set Object = object.Add( [ Name [, Caption [, index]]])
```

For other controls

```
Set Control = object.Add( ProgID [, Name [, Visible]])
```

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

| Part           | Description   |
|----------------|---|
| <i>object</i>  | Required. A valid object name.  |
| <i>Name</i>    | Optional. Specifies the name of the object being added. If a name is not specified, the system generates a default name based on the rules of the application where the HTML Layout is used.  |
| <i>Caption</i> | Optional. Specifies the caption to appear on a tab or a control. If a caption is not specified, the system generates a default caption based on the rules of the application where the HTML Layout is used.                                 |
| <i>index</i>   | Optional. Identifies the position of a tab within a <b>Tabs</b> collection. If an index is not specified, the system appends the tab to the end of the <b>Tabs</b> collection and assigns the appropriate index value.                      |
| <i>ProgID</i>  | Required. Programmatic identifier. A text string with no spaces that identifies an object class. The standard syntax of a <i>ProgID</i> is <Vendor>.<Component>.<Version>. A <i>ProgID</i> is mapped to a <u>class identifier (CLSID)</u> . |
| <i>Visible</i> | Optional. <b>True</b> if the object is visible (default). <b>False</b> if the object is hidden.   |

### Settings

*ProgID* values for individual controls are:

| Control              | ProgID value          |
|----------------------|-----------------------|
| <b>CheckBox</b>      | Forms.CheckBox.1      |
| <b>ComboBox</b>      | Forms.ComboBox.1      |
| <b>CommandButton</b> | Forms.CommandButton.1 |
| <b>Image</b>         | Forms.Image.1         |
| <b>Label</b>         | Forms.Label.1         |
| <b>ListBox</b>       | Forms.ListBox.1       |
| <b>OptionButton</b>  | Forms.OptionButton.1  |
| <b>ScrollBar</b>     | Forms.ScrollBar.1     |
| <b>SpinButton</b>    | Forms.SpinButton.1    |
| <b>TabStrip</b>      | Forms.TabStrip.1      |
| <b>TextBox</b>       | Forms.TextBox.1       |

## ToggleButton

Forms.ToggleButton.1

### Remarks

For a **TabStrip**, the **Add** method returns a **Tab** object. The index value for the first **Tab** of a collection is 0, the value for the second **Tab** is 1, and so on.

For the **Controls** collection of an object, the **Add** method returns a control corresponding to the specified *ProgID*.

The following syntax will return the **Text** property of a control added at design time:

```
userform1.thebox.text
```

If you add a control a run time, you must use the exclamation syntax to reference properties of that control. For example, to return the **Text** property of a control added at run time, use the following syntax:

```
userform1!thebox.text
```

**Note** You can change a control's **ID** property at run time only if you added that control at run time with the **Add** method.

# AddItem Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3mthAddItemC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3mthAddItemX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3mthAddItemA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3mthAddItemS"}

For a single-column list box or combo box, adds an item to the list. For a multicolumn list box or combo box, adds a row to the list.

## Syntax

*Variant* = *object*.**AddItem**( [ *item* [, *varIndex*]])

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

| Part            | Description   |
|-----------------|---|
| <i>object</i>   | Required. A valid object.   |
| <i>item</i>     | Optional. Specifies the item or row to add. The number of the first item or row is 0; the number of the second item or row is 1, and so on. |
| <i>varIndex</i> | Optional. Integer specifying the position within the object where the new item or row is placed.  |

## Remarks

If you supply a valid value for *varIndex*, the **AddItem** method places the item or row at that position within the list. If you omit *varIndex*, the method adds the item or row at the end of the list.

The value of *varIndex* must not be greater than the value of the **ListCount** property.

For a multicolumn **ListBox** or **ComboBox**, **AddItem** inserts an entire row; that is, it inserts an item for each column of the control. To assign values to an item beyond the first column, use the **List** or **Column** property and specify the row and column of the item.

**Note** You can add more than one row at a time to a **ComboBox** or **ListBox** by using **List**.



## Clear Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3mthClearC"} {ewc HLP95EN.DLL, DYNALINK,  
"Example":"f3mthClearX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3mthClearA"} {ewc  
HLP95EN.DLL, DYNALINK, "Specifics":"f3mthClearS"}

Removes all objects from an object or collection.

### Syntax

*object*.**Clear**

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

| Part          | Description               |
|---------------|---------------------------|
| <i>object</i> | Required. A valid object. |

### Remarks

For a **TabStrip**, the **Clear** method deletes individual tabs.

For a **ListBox** or **ComboBox**, **Clear** removes all entries in the list.

For a **Controls** collection, **Clear** deletes controls that were created at run time with the **Add** method. Using **Clear** on controls created at design time causes an error.

## Copy Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3mthCopyC"} {ewc HLP95EN.DLL, DYNALINK,  
"Example":"f3mthCopyX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3mthCopyA"} {ewc  
HLP95EN.DLL, DYNALINK, "Specifics":"f3mthCopyS"}

Copies the contents of an object to the Clipboard.

### Syntax

*object*.Copy

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

| Part          | Description               |
|---------------|---------------------------|
| <i>object</i> | Required. A valid object. |

### Remarks

The original content remains on the object.

The actual content that is copied depends on the object. For example, On a **TextBox** or **ComboBox**, the **Copy** method copies the currently selected text.

Using **Copy** for an HTML Layout copies the currently active control.

## Cut Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3mthCutC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3mthCutX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3mthCutA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3mthCutS"}

Removes selected information from an object and transfers it to the Clipboard.

### Syntax

*object*.Cut

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

| Part          | Description               |
|---------------|---------------------------|
| <i>object</i> | Required. A valid object. |

### Remarks

For a **ComboBox** or **TextBox**, the **Cut** method removes currently selected text in the control to the Clipboard. This method does not require that the control have the focus.

On an HTML Layout, **Cut** removes currently selected controls to the Clipboard. This method only removes controls created at run time.

## DropDown Method

```
{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3mthDropDownC"} {ewc HLP95EN.DLL, DYNALINK,  
"Example":"f3mthDropDownX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3mthDropDownA"} {ewc  
HLP95EN.DLL, DYNALINK, "Specifics":"f3mthDropDownS"}
```

Displays the list portion of a **ComboBox**; or, if the list is currently displayed, dismisses it.

### Syntax

*object*.**DropDown**

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

| Part          | Description               |
|---------------|---------------------------|
| <i>object</i> | Required. A valid object. |

### Remarks

Use the **DropDown** method to open the list in a combo box.

# GetFormat Method

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3mthGetFormatC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3mthGetFormatX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3mthGetFormatA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3mthGetFormatS"}

Returns an integer value indicating whether a specific format is on the **DataObject**.

## Syntax

*Boolean* = *object*.**GetFormat**( [ *format* ])

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

| Part          | Description   |
|---------------|---|
| <i>object</i> | Required. A valid object.   |
| <i>format</i> | Optional. An integer or string specifying the data format to use when pasting the Clipboard contents. |

## Settings

The settings for *format* are:

| Constant        | Value | Description  |
|-----------------|-------|--------------|
| <i>fmCFText</i> | 1     | Text format. |

## Remarks

The **GetFormat** method searches for a format in the current list of formats on the **DataObject**. If the format is on the **DataObject**, **GetFormat** returns 1; if not, **GetFormat** returns 0.

The **DataObject** currently supports only text formats.

# GetFromClipboard, GetText Methods

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3mthGetFromClipboardC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3mthGetFromClipboardX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3mthGetFromClipboardA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3mthGetFromClipboardS"}

**GetFromClipboard** moves data from the Clipboard to a **DataObject**. **GetText** retrieves a text string from the Clipboard using a specified format.

## Syntax

*String* = *object*.**GetFromClipboard**( [ *format* ])

*String* = *object*.**GetText**( [ *format* ])

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

| Part          | Description   |
|---------------|---|
| <i>object</i> | Required. A valid object name.  |
| <i>format</i> | Optional. An integer specifying the data format to use when pasting the Clipboard contents. |

## Settings

The settings for *format* are:

| Constant        | Value | Description  |
|-----------------|-------|--------------|
| <i>fmCFText</i> | 1     | Text format. |

## Remarks

The **DataObject** and the Clipboard support multiple formats, but only support one data item of a given format. For example, the **DataObject** might include one text item, but cannot include two text items of the type **fmCFText**.

If the **DataObject** contains data in the same format as new data, the new data replaces the existing data in the **DataObject**. If the new data is in a new format, the new data and the new format are both added to the **DataObject**.

If no format is specified, the **GetText** method returns the string associated with the standard text format.

## Item Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3mthItemC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3mthItemX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3mthItemA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3mthItemS"}

Returns a member of a collection, either by position or by name.

### Syntax

**Set** *Object* = *object*.**Item**( *collectionindex* )

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

| <b>Part</b>            | <b>Description</b>  |
|------------------------|---|
| <i>object</i>          | Required. A valid object.                                     |
| <i>collectionindex</i> | Required. A member's position, or index, within a collection. |

### Settings

The *collectionindex* can be either a string or an integer. If it is a string, it must be a valid member name. If it is an integer, the minimum value is 0 and the maximum value is one less than the number of items in the collection.

### Remarks

If an invalid index or name is specified, an error occurs.

## Move Method

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3mthMoveC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3mthMoveX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3mthMoveA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3mthMoveS"}

Moves an HTML Layout or control, or moves all the controls in the **Controls** collection.

### Syntax

*object*.**Move**( [*Left* [, *Top* [, *Width* [, *Height* ]]] )

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

| <b>Part</b>   | <b>Description</b>   |
|---------------|--|
| <i>object</i> | Required. A valid object name.   |
| <i>Left</i>   | Optional. <u>Single-precision value</u> , in points, indicating the horizontal coordinate for the left edge of the object. |
| <i>Top</i>    | Optional. Single-precision value, in points, that specifies the vertical coordinate for the top edge of the object.        |
| <i>Width</i>  | Optional. Single-precision value, in points, indicating the width of the object.   |
| <i>Height</i> | Optional. Single-precision value, in points, indicating the height of the object.  |

### Settings

### Remarks

For an HTML Layout or control, you can move a selection to a specific location relative to the edges of the HTML Layout that contains the selection.

You can use named arguments, or you can enter the arguments by position. If you use named arguments, you can list the arguments in any order. If not, you must enter the arguments in the order shown, using commas to indicate the relative position of arguments you do not specify. Any unspecified arguments remain unchanged.



## Paste Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3mthPasteC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3mthPasteX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3mthPasteA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3mthPasteS"}

Transfers the contents of the Clipboard to an object.

### Syntax

*object*.Paste

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

| Part          | Description               |
|---------------|---------------------------|
| <i>object</i> | Required. A valid object. |

### Remarks

Data pasted into a **ComboBox** or **TextBox** is treated as text.

When the **Paste** method is used with an HTML Layout, you can paste any object onto the HTML Layout.

## PutInClipboard Method

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3mthPutInClipboardC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3mthPutInClipboardX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3mthPutInClipboardA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3mthPutInClipboardS"}

Moves data from a **DataObject** to the Clipboard.

### Syntax

*object*.PutInClipboard

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

| Part          | Description               |
|---------------|---------------------------|
| <i>object</i> | Required. A valid object. |

### Remarks

The **DataObject** and the Clipboard both support multiple formats, but only support one data item of a given format. For example, the **DataObject** might include one text item stored using the Clipboard format.

If the Clipboard contains data in the same format as new data, the new data replaces the existing data on the Clipboard. If the new data is in a new format, the new data and the new format are both added to the Clipboard.

## Remove Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3mthRemoveC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3mthRemoveX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3mthRemoveA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3mthRemoveS"}

Removes a member from a collection; or, removes a control from an HTML Layout.

### Syntax

*object*.**Remove**( *collectionindex*)

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

| Part                   | Description  |
|------------------------|--|
| <i>object</i>          | Required. A valid object.  |
| <i>collectionindex</i> | Required. A member's position, or index, within a collection. Numeric as well as string values are acceptable. If the value is a number, the minimum value is zero, and the maximum value is one less than the number of members in the collection. If the value is a string, it must correspond to a valid member name. |

### Remarks

This method deletes any control that was added at run time. However, attempting to delete a control that was added at design time will result in an error.

## RemoveItem Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3mthRemoveItemC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3mthRemoveItemX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3mthRemoveItemA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3mthRemoveItemS"}

Removes a row from the list in a list box or combo box.

### Syntax

*Boolean* = *object*.**RemoveItem**( *index*)

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

| Part          | Description  |
|---------------|--|
| <i>object</i> | Required. A valid object.  |
| <i>index</i>  | Required. Specifies the row to delete. The number of the first row is 0; the number of the second row is 1, and so on. |

## SetFocus Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3mthSetFocusC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3mthSetFocusX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3mthSetFocusA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3mthSetFocusS"}

Moves the focus to this instance of an object.

### Syntax

*object*.**SetFocus**

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

| Part          | Description               |
|---------------|---------------------------|
| <i>object</i> | Required. A valid object. |

### Remarks

If setting the focus fails, the focus reverts to the previous object and an error is generated.

By default, setting the focus to a control does not activate the control's window or place it on top of other controls.

## SetText Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3mthSetTextC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3mthSetTextX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3mthSetTextA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3mthSetTextS"}

Copies a text string to the Clipboard using a specified format.

### Syntax

*object*.**SetText**( *StoreData* [, *format*])

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

| Part             | Description   |
|------------------|---|
| <i>object</i>    | Required. A valid object.   |
| <i>StoreData</i> | Required. Defines the data to store on the Clipboard.   |
| <i>format</i>    | Optional. An integer or string specifying the data format to use when pasting the Clipboard contents. |

### Settings

The settings for *format* are:

| Constant        | Value | Description  |
|-----------------|-------|--------------|
| <i>fmCFText</i> | 1     | Text format. |

### Remarks

The Clipboard stores data according to its format. When the user supplies a string, the Clipboard saves the text under the specified format.

If no format is specified, the **SetText** method assigns the standard text format to the text string. If a new format is specified, the Clipboard registers the new format with the system.

## StartDrag Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3mthStartDragC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3mthStartDragX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3mthStartDragA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3mthStartDragS"}

Initiates a drag-and-drop operation for a **DataObject**.

### Syntax

*fmDropEffect*=**Object.StartDrag**([*Effect* as *fmDropEffect*])

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

| Part | Description |
|------|-------------|
|------|-------------|

|               |                           |
|---------------|---------------------------|
| <i>Object</i> | Required. A valid object. |
|---------------|---------------------------|

|               |   |
|---------------|---|
| <i>Effect</i> | Optional. Effect of the drop operation on the target control. |
|---------------|---|

### Settings

The settings for *Effect* are:

| Constant                      | Value | Description  |
|-------------------------------|-------|--|
| <i>fmDropEffectNone</i>       | 0     | Does not copy or move the <u>drop source</u> to the drop target. |
| <i>fmDropEffectCopy</i>       | 1     | Copies the drop source to the drop target.                       |
| <i>fmDropEffectMove</i>       | 2     | Moves the drop source to the drop target.                        |
| <i>fmDropEffectCopyOrMove</i> | 3     | Copies or moves the drop source to the drop target.              |

### Remarks

The drag action starts at the current mouse pointer position with the current keyboard state and ends when the user releases the mouse. The effect of the drag-and-drop operation depends on the effect chosen for the drop target.

For example, a control's MouseMove event might include the **StartDrag** method. When the user clicks the control and moves the mouse, the mouse pointer changes to indicate whether *Effect* is valid for the drop target.

## ZOrder Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3mthZOrderC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3mthZOrderX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3mthZOrderA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3mthZOrderS"}

Places the object at the front or back of the z-order.

### Syntax

*object*.ZOrder( [ *zPosition*])

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

| Part             | Description  |
|------------------|--|
| <i>object</i>    | Required. A valid object.  |
| <i>zPosition</i> | Optional. A control's position, front or back, in the container's z-order. |

### Settings

The settings for *zPosition* are:

| Constant        | Value | Description   |
|-----------------|-------|---|
| <i>fmTop</i>    | 0     | Places the control at the front of the z-order. The control appears on top of other controls (default). |
| <i>fmBottom</i> | 1     | Places the control at the back of the z-order. The control appears underneath other controls.           |

### Remarks

The z-order determines how windows and controls are stacked when they are presented to the user. Items at the back of the z-order are overlaid by closer items; items at the front of the z-order appear to be on top of items at the back. When the *zPosition* argument is omitted, the object is brought to the front.

In design time, the Bring to Front or Send To Back commands set the z-order. Bring to Front is equivalent to using the **ZOrder** method and putting the object at the front of the z-order. Send to Back is equivalent to using **ZOrder** and putting the object at the back of the z-order.

This method does not affect content or sequence of the controls in the **Controls** collection.

**Note** You can't Undo or Redo layering commands such as **Send to Back** or **Bring to Front**. For example, if you select an object and click **Move Backward** on the shortcut menu, you won't be able to Undo or redo that action.



# Accelerator Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proAcceleratorC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proAcceleratorX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proAcceleratorA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proAcceleratorS"}

Sets or retrieves the accelerator key for a control.

## Syntax

*object*.**Accelerator** [= *String*]

The **Accelerator** property syntax has these parts:

| Part          | Description  |
|---------------|--|
| <i>object</i> | Required. A valid object.                              |
| <i>String</i> | Optional. The character to use as the accelerator key. |

## Remarks

To designate an accelerator key, enter a single character for the **Accelerator** property. You can set **Accelerator** in the control's Properties window or in code. If the value of this property contains more than one character, the first character in the string becomes the value of **Accelerator**.

When an accelerator key is used, there is no visual feedback (other than focus) to indicate that the control initiated the Click event. For example, if the accelerator key applies to a **CommandButton**, the user will not see the button pressed in the interface. The button receives the focus, however, when the user presses the accelerator key.

If the accelerator applies to a **Label**, the control following the **Label** in the tab order, rather than the **Label** itself, receives the focus.

## ActiveControl Property

```
{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proActiveControlC"}           {ewc HLP95EN.DLL, DYNALINK,  
"Example":"f3proActiveControlX":1}           {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proActiveControlA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proActiveControlS"}
```

Identifies and allows manipulation of the control that has the focus.

### Syntax

*object*.**ActiveControl**

The **ActiveControl** property syntax has these parts:

| Part          | Description               |
|---------------|---------------------------|
| <i>object</i> | Required. A valid object. |

### Remarks

The **ActiveControl** property is read-only and is set when you select a control in the interface. You can use **ActiveControl** as a substitute for the control name when setting properties or calling methods.

# Alignment Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proAlignmentC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proAlignmentX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proAlignmentA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proAlignmentsS"}

Specifies the position of a control relative to its caption.

## Syntax

*object*.**Alignment** [= *fmAlignment*]

The **Alignment** property syntax has these parts:

| Part               | Description                 |
|--------------------|-----------------------------|
| <i>object</i>      | Required. A valid object.   |
| <i>fmAlignment</i> | Optional. Caption position. |

## Settings

The settings for *fmAlignment* are:

| Constant                | Value | Description   |
|-------------------------|-------|---|
| <i>fmAlignmentLeft</i>  | 0     | Places the caption to the left of the control.            |
| <i>fmAlignmentRight</i> | 1     | Places the caption to the right of the control (default). |

## Remarks

The caption text for a control is left-aligned.

**Note** Although the **Alignment** property exists on the **ToggleButton**, the property is disabled. You cannot set or return a value for this property on the **ToggleButton**.

# AutoSize Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proAutoSizeC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proAutoSizeX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proAutoSizeA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proAutoSizeS"}

Specifies whether an object automatically resizes to display its entire contents.

## Syntax

*object*.**AutoSize** [= *Boolean*]

The **AutoSize** property syntax has these parts:

| Part           | Description   |
|----------------|---|
| <i>object</i>  | Required. A valid object.                           |
| <i>Boolean</i> | Optional. Specifies whether the control is resized. |

## Settings

The settings for *Boolean* are:

| Value        | Description  |
|--------------|--|
| <b>True</b>  | Automatically resizes the control to display its entire contents.  |
| <b>False</b> | Keeps the size of the control constant. Contents are cropped when they exceed the area of the control (default). |

## Remarks

For controls with captions, the **AutoSize** property specifies whether the control automatically adjusts to display the entire caption.

For controls without captions, this property specifies whether the control automatically adjusts to display the information stored in the control. In a **ComboBox**, for example, setting **AutoSize** to **True** automatically sets the width of the display area to match the length of the current text.

For a single-line text box, setting **AutoSize** to **True** automatically sets the width of the display area to the length of the text in the text box.

For a multiline text box that contains no text, setting **AutoSize** to **True** automatically displays the text as a column. The width of the text column is set to accommodate the widest letter of that font size. The height of the text column is set to display the entire text of the **TextBox**.

For a multiline text box that contains text, setting **AutoSize** to **True** automatically enlarges the **TextBox** vertically to display the entire text. The width of the **TextBox** does not change.

**Note** If you manually change the size of a control while **AutoSize** is **True**, the manual change overrides the size previously set by **AutoSize**.

# AutoTab Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proAutoTabC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proAutoTabX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proAutoTabA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proAutoTabS"}

Specifies whether an automatic tab occurs when a user enters the maximum allowable number of characters into a **TextBox** or the text box portion of a **ComboBox**.

## Syntax

*object*.**AutoTab** [= *Boolean*]

The **AutoTab** property syntax has these parts:

| Part           | Description  |
|----------------|--|
| <i>object</i>  | Required. A valid object.                            |
| <i>Boolean</i> | Optional. Specifies whether an automatic tab occurs. |

## Settings

The settings for *Boolean* are:

| Value        | Description                   |
|--------------|-------------------------------|
| <b>True</b>  | Tab occurs.                   |
| <b>False</b> | Tab does not occur (default). |

## Remarks

The **MaxLength** property specifies the maximum number of characters allowed in a **TextBox** or the text box portion of a **ComboBox**.

You can specify the **AutoTab** property for a **TextBox** or **ComboBox** on an HTML Layout for which you usually enter a set number of characters. Once a user enters the maximum number of characters, the focus automatically moves to the next control in the tab order. For example, if a **TextBox** displays inventory stock numbers that are always five characters long, you can use **MaxLength** to specify the maximum number of characters to enter into the **TextBox** and **AutoTab** to automatically tab to the next control after the user enters five characters.

Support for AutoTab varies from one application to another. Not all containers support this property.

# AutoWordSelect Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proAutoWordSelectC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proAutoWordSelectX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proAutoWordSelectA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proAutoWordSelectS"}

Specifies whether a word or a character is the basic unit used to extend a selection.

## Syntax

*object*.**AutoWordSelect** [= *Boolean*]

The **AutoWordSelect** property syntax has these parts:

| Part           | Description  |
|----------------|--|
| <i>object</i>  | Required. A valid object.                                      |
| <i>Boolean</i> | Optional. Specifies the basic unit used to extend a selection. |

## Settings

The settings for *Boolean* are:

| Value        | Description                              |
|--------------|--|
| <b>True</b>  | Uses a word as the basic unit (default). |
| <b>False</b> | Uses a character as the basic unit.      |

## Remarks

The **AutoWordSelect** property specifies how the selection extends or contracts in the edit region of a **TextBox** or **ComboBox**.

If the user places the insertion point in the middle of a word and then extends the selection while **AutoWordSelect** is **True**, the selection includes the entire word.

# BackColor Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proBackColorC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proBackColorX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proBackColorA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proBackColorS"}

Specifies the background color of the object.

## Syntax

*object*.**BackColor** [= *Long*]

The **BackColor** property syntax has these parts:

| Part          | Description  |
|---------------|--|
| <i>object</i> | Required. A valid object.  |
| <i>Long</i>   | Optional. A value or constant that determines the background color of an object. |

## Settings

You can use any integer that represents a valid color. You can also specify a color by using the RGB function with red, green, and blue color components. The value of each color component is an integer that ranges from zero to 255. For example, you can specify teal blue as the integer value 4966415 or as red, green, and blue color components 15, 200, 75.

## Remarks

You can see the background color of an object only if the **BackStyle** property is set to **fmBackStyleOpaque**.

# BackStyle Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proBackStyleC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proBackStyleX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proBackStyleA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proBackStyleS"}

Returns or sets the background style for an object.

## Syntax

*object*.**BackStyle** [= *fmBackStyle*]

The **BackStyle** property syntax has these parts:

| Part               | Description                                 |
|--------------------|---|
| <i>object</i>      | Required. A valid object.                   |
| <i>fmBackStyle</i> | Optional. Specifies the control background. |

## Settings

The settings for *fmBackStyle* are:

| Constant                      | Value | Description                         |
|-------------------------------|-------|-------------------------------------|
| <i>fmBackStyleTransparent</i> | 0     | The background is transparent.      |
| <i>fmBackStyleOpaque</i>      | 1     | The background is opaque (default). |

## Remarks

The **BackStyle** property determines whether a control is transparent. If **BackStyle** is **fmBackStyleOpaque**, the control is not transparent and you cannot see anything behind the control on an HTML Layout. If **BackStyle** is **fmBackStyleTransparent**, you can see through the control and look at anything on the HTML Layout located behind the control.

**Note** **BackStyle** does not affect the transparency of bitmaps. You must use a picture editor to make a bitmap transparent. Not all controls support transparent bitmaps.



# Bold, Italic, Size, StrikeThrough, Underline, Weight Properties

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proBoldC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proBoldX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proBoldA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proBoldS"}

Specifies the visual attributes of text on a displayed or printed HTML Layout.

## Syntax

*object*.**Bold** [= *Boolean*]  
*object*.**Italic** [= *Boolean*]  
*object*.**Size** [= *Currency*]  
*object*.**StrikeThrough** [= *Boolean*]  
*object*.**Underline** [= *Boolean*]  
*object*.**Weight** [= *Integer*]

The **Bold**, **Italic**, **Size**, **StrikeThrough**, **Underline**, and **Weight** property syntaxes have these parts:

| Part            | Description                                  |
|-----------------|--|
| <i>object</i>   | Required. A valid object name.               |
| <i>Boolean</i>  | Optional. Specifies the font style.          |
| <i>Currency</i> | Optional. A number indicating the font size. |
| <i>Integer</i>  | Optional. Specifies the font style.          |

The settings for *Boolean* are:

| Value        | Description   |
|--------------|---|
| <b>True</b>  | The text has the specified attribute (that is bold, italic, size, strikethrough or underline marks, or weight). |
| <b>False</b> | The text does not have the specified attribute (default).   |

The **Weight** property accepts values from 0 to 1000. A value of zero allows the system to pick the most appropriate weight. A value from 1 to 1000 indicates a specific weight, where 1 represents the lightest type and 1000 represents the darkest type.

## Remarks

These properties define the visual characteristics of text. The **Bold** property determines whether text is normal or bold. The **Italic** property determines whether text is normal or italic. The **Size** property determines the height, in points, of displayed text. The **Underline** property determines whether text is underlined. The **StrikeThrough** property determines whether the text appears with strikethrough marks. The **Weight** property determines the darkness of the type.

The font's appearance on screen and in print may differ, depending on your computer and printer. If you select a font that your system can't display with the specified attribute or that isn't installed, Windows substitutes a similar font. The substitute font will be as similar as possible to the font originally requested.

Changing the value of **Bold** also changes the value of **Weight**. Setting **Bold** to **True** sets **Weight** to 700; setting **Bold** to **False** sets **Weight** to 400. Conversely, setting **Weight** to anything over 550 sets **Bold** to **True**; setting **Weight** to 550 or less sets **Bold** to **False**.

The default point size is determined by the operating system.

## BorderColor Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proBorderColorC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proBorderColorX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proBorderColorA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proBorderColorS"}

Specifies the color of a control's border.

### Syntax

*object*.**BorderColor** [= *Long*]

The **BorderColor** property syntax has these parts:

| <b>Part</b>   | <b>Description</b>   |
|---------------|--|
| <i>object</i> | Required. A valid object.  |
| <i>Long</i>   | Optional. A value or constant that determines the border color of a control. |

### Settings

You can use any integer that represents a valid color. You can also specify a color by using the [RGB](#) function with red, green, and blue color components. The value of each color component is an integer that ranges from zero to 255. For example, you can specify teal blue as the integer value 4966415 or as RGB color component values 15, 200, 75.

### Remarks

To use the **BorderColor** property, the **BorderStyle** property must be set to a value other than **fmBorderStyleNone**.

**BorderStyle** uses **BorderColor** to define the border colors. The **SpecialEffects** property uses [system colors](#) exclusively to define its border colors. For Windows operating systems, System Color settings are stored in the Control Panel, either in the Desktop folder or the Color folder.

# BorderStyle Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proBorderStyleC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proBorderStyleX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proBorderStyleA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proBorderStyleS"}

Specifies the type of border used by a control.

## Syntax

*object*.**BorderStyle** [= *fmBorderStyle*]

The **BorderStyle** property syntax has these parts:

| Part                 | Description                           |
|----------------------|---------------------------------------|
| <i>object</i>        | Required. A valid object.             |
| <i>fmBorderStyle</i> | Optional. Specifies the border style. |

## Settings

The settings for *fmBorderStyle* are:

| Constant                   | Value | Description                                     |
|----------------------------|-------|---|
| <i>fmBorderStyleNone</i>   | 0     | The control has no visible border line.         |
| <i>fmBorderStyleSingle</i> | 1     | The control has a single-line border (default). |

The default value for a **ComboBox**, **Label**, **ListBox** or **TextBox** is 0 (*None*). The default value for an **Image** is 1 (*Single*).

## Remarks

You can use either **BorderStyle** or **SpecialEffect** to specify the border for a control, but not both. If you specify a nonzero value for one of these properties, the system sets the value of the other property to zero. For example, if you set **BorderStyle** to **fmBorderStyleSingle**, the system sets **SpecialEffect** to zero (*Flat*). If you specify a nonzero value for **SpecialEffect**, the system sets **BorderStyle** to zero.

**BorderStyle** uses **BorderColor** to define the colors of its borders.

## BoundColumn Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proBoundColumnC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proBoundColumnX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proBoundColumnA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proBoundColumnS"}

Identifies the source of data in a multicolumn **ComboBox** or **ListBox**.

### Syntax

*object*.**BoundColumn** [= *Variant*]

The **BoundColumn** property syntax has these parts:

| Part           | Description   |
|----------------|---|
| <i>object</i>  | Required. A valid object.   |
| <i>Variant</i> | Optional. Indicates how the <b>BoundColumn</b> value is selected. |

### Settings

The settings for *Variant* are:

| Value        | Description   |
|--------------|---|
| 0            | Assigns the value of the <b>ListIndex</b> property to the control.  |
| 1 or greater | Assigns the value from the specified column to the control. Columns are numbered from 1 when using this property (default). |

### Remarks

When the user chooses a row in a multicolumn **ListBox** or **ComboBox**, the **BoundColumn** property identifies which item from that row to store as the value of the control. For example, if each row contains 8 items and **BoundColumn** is 3, the system stores the information in the third column of the currently-selected row as the value of the object.

You can display one set of data to users but store different, associated values for the object by using the **BoundColumn** and the **TextColumn** properties. **TextColumn** identifies the column of data displayed in a **ComboBox** or **ListBox**; **BoundColumn** identifies the column of associated data values stored for the control. For example, you could set up a multicolumn **ListBox** that contains the names of holidays in one column and dates for the holidays in a second column. To present the holiday names to users, specify the first column as the **TextColumn**. To store the dates of the holidays, specify the second column as the **BoundColumn**.

The **ListIndex** value retrieves the number of the selected row. For example, if you want to know the row of the selected item, set **BoundColumn** to 0 to assign the number of the selected row as the value of the control. Be sure to retrieve a current value, rather than relying on a previously saved value, if you are referencing a list whose contents might change.

The **Column**, **List**, and **ListIndex** properties all use zero-based numbering. That is, the value of the first item (column or row) is zero; the value of the second item is one, and so on. This means that if **BoundColumn** is set to 3, you could access the value stored in that column using the expression **Column(2)**.

## CanPaste Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proCanPasteC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proCanPasteX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proCanPasteA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proCanPasteS"}

Specifies whether the Clipboard contains data that the object supports.

### Syntax

*object*.**CanPaste** [=*Boolean*]

The **CanPaste** property syntax has these parts:

| Part           | Description   |
|----------------|---|
| <i>object</i>  | Required. A valid object.                                       |
| <i>Boolean</i> | Optional. Specifies whether a paste operation can be performed. |

### Return Values

The **CanPaste** property return values are:

| Value        | Description  |
|--------------|--|
| <b>True</b>  | The object underneath the mouse pointer can receive information pasted from the Clipboard (default). |
| <b>False</b> | The object underneath the mouse pointer cannot receive information pasted from the Clipboard.        |

### Remarks

If the Clipboard data is in a format that the current target object does not support, the **CanPaste** property is **False**. For example, if you try to paste a bitmap into an object that only supports text, **CanPaste** will be **False**.

## Caption Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proCaptionC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proCaptionX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proCaptionA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proCaptionS"}

Descriptive text that appears on an object to identify or describe it.

### Syntax

*object*.**Caption** [= *String*]

The **Caption** property syntax has these parts:

| Part          | Description  |
|---------------|--|
| <i>object</i> | Required. A valid object.  |
| <i>String</i> | Optional. A string expression that evaluates to the text displayed as the caption. |

### Settings

The default setting for a control is a unique name based on the type of control. For example, `CommandButton1` is the default caption for the first command button in an HTML Layout.

### Remarks

The text identifies or describes the object with which it is associated. For buttons and labels, the **Caption** property specifies the text that appears in the control. For **Tab** objects, it specifies the text that appears on the tab.

If a control's caption is too long, the caption is truncated. If an HTML Layout's caption is too long for the title bar, the title is displayed with an ellipsis.

The **ForeColor** property of the control determines the color of the text in the caption.

**Tip** If a control has both the **Caption** and **AutoSize** properties, setting **AutoSize** to **True** automatically adjusts the size of the control to frame the entire caption.

## ClientHeight, ClientLeft, ClientTop, ClientWidth Properties

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proClientHeightC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proClientHeightX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proClientHeightA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proClientHeightS"}

Define the dimensions and location of the display area of a **TabStrip**.

### Syntax

*object*.**ClientHeight** [=Single]

*object*.**ClientLeft** [=Single]

*object*.**ClientTop** [=Single]

*object*.**ClientWidth** [=Single]

The **ClientHeight**, **ClientLeft**, **ClientTop**, and **ClientWidth** property syntaxes have these parts:

| Part          | Description   |
|---------------|---|
| <i>object</i> | Required. A valid object.   |
| <i>Single</i> | Optional. For <b>ClientHeight</b> and <b>ClientWidth</b> , specifies the height or width, in points, of the display area. For <b>ClientLeft</b> and <b>ClientTop</b> , specifies the distance, in points, from the top or left edge of the <b>TabStrip's</b> container. |

### Remarks

At run time, **ClientLeft**, **ClientTop**, **ClientHeight**, and **ClientWidth** automatically store the coordinates and dimensions of the **TabStrip's** internal area, which is shared by objects in the **TabStrip**.

# Column Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proColumnC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proColumnX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proColumnA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proColumnS"}

Specifies one or more items in a **ListBox** or **ComboBox**.

## Syntax

*object*.**Column**( *column*, *row* ) [= *Variant*]

The **Column** property syntax has these parts:

| Part           | Description  |
|----------------|--|
| <i>object</i>  | Required. A valid object.  |
| <i>column</i>  | Optional. An integer with a range from 0 to one less than the total number of columns.   |
| <i>row</i>     | Optional. An integer with a range from 0 to one less than the total number of rows.  |
| <i>Variant</i> | Optional. Specifies a single value, a column of values, or a two-dimensional <u>array</u> to load into a <b>ListBox</b> or <b>ComboBox</b> . |

## Settings

If you specify both the column and row values, **Column** reads or writes a specific item.

If you specify only the column value, the **Column** property reads or writes the specified column in the current row of the object. For example, MyListBox.Column (3) reads or writes the third column in MyListBox.

**Column** returns a *Variant* from the cursor. When a built-in cursor provides the value for *Variant* (such as when using the **AddItem** method), the value is a string. When an external cursor provides the value for *Variant*, formatting associated with the data is not included in the *Variant*.

## Remarks

You can use **Column** to assign the contents of a combo box or list box to another control, such as a text box.

If the user makes no selection when you refer to a column in a combo box or list box, the **Column** setting is **Null**. You can check for this condition by using the **IsNull** function.

You can also use **Column** to copy an entire two-dimensional array of values to a control. This syntax lets you quickly load a list of choices rather than individually loading each element of the list using **AddItem**.

**Note** When copying data from a two-dimensional array, **Column** transposes the contents of the array in the control so that the contents of ListBox1.Column(X, Y) is the same as MyArray(Y, X). You can also use **List** to copy an array without transposing it.



## ColumnCount Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proColumnCountC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proColumnCountX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proColumnCountA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proColumnCountS"}

Specifies the number of columns to display in a list box or combo box.

### Syntax

*object*.**ColumnCount** [= *Long*]

The **ColumnCount** property syntax has these parts:

| Part          | Description   |
|---------------|---|
| <i>object</i> | Required. A valid object.                             |
| <i>Long</i>   | Optional. Specifies the number of columns to display. |

### Remarks

If you set the **ColumnCount** property for a list box to 3 on an employee form, one column can list last names, another can list first names, and the third can list employee ID numbers.

Setting **ColumnCount** to 0 displays zero columns, and setting it to -1 displays all the available columns. There is a 10-column limit (0 to 9).

You can use the **ColumnWidths** property to set the width of the columns displayed in the control.

## ColumnHeads Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proColumnHeadsC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proColumnHeadsX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proColumnHeadsA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proColumnHeadsS"}

Displays a single row of column headings for list boxes, combo boxes, and objects that accept column headings.

### Syntax

*object*.**ColumnHeads** [= *Boolean*]

The **ColumnHeads** property syntax has these parts:

| Part           | Description  |
|----------------|--|
| <i>object</i>  | Required. A valid object.                                      |
| <i>Boolean</i> | Optional. Specifies whether the column headings are displayed. |

### Settings

The settings for *Boolean* are:

| Value        | Description                               |
|--------------|---|
| <b>True</b>  | Display column headings.                  |
| <b>False</b> | Do not display column headings (default). |

Headings in combo boxes appear only when the list drops down.

### Remarks

When the system uses the first row of data items as column headings, they can't be selected.

# ColumnWidths Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proColumnWidthsC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proColumnWidthsX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proColumnWidthsA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proColumnWidthsS"}

Specifies the width of each column in a multicolumn combo box or list box.

## Syntax

*object*.**ColumnWidths** [= *String*]

The **ColumnWidths** property syntax has these parts:

| Part          | Description   |
|---------------|---|
| <i>object</i> | Required. A valid object.   |
| <i>String</i> | Optional. Sets the column width in points. A setting of -1 or blank results in a calculated width. A width of 0 hides a column. To specify a different unit of measurement, include the unit of measure. A value greater than 0 explicitly specifies the width of the column. |

## Settings

To separate column entries, use semicolons (;) as list separators. Or use the list separator specified in the Regional Settings section of the Windows Control Panel.

Any or all of the **ColumnWidths** property settings can be blank. You create a blank setting by typing a list separator without a preceding value.

If you specify a -1 in the property page, the displayed value in the property page is a blank.

To calculate column widths when **ColumnWidths** is blank or -1, the width of the control is divided equally among all columns of the list. If the sum of the specified column widths exceeds the width of the control, the list is left-aligned within the control and one or more of the rightmost columns are not displayed. Users can scroll the list using the horizontal scroll bar to display the rightmost columns.

The minimum calculated column width is 72 points (1 inch). To produce columns narrower than this, you must specify the width explicitly.

Unless specified otherwise, column widths are measured in points. To specify another unit of measure, include the units as part of the values. The following examples specify column widths in several units of measure and describe how the various settings would fit in a three-column list box that is 4 inches wide.

| Setting         | Effect  |
|-----------------|---|
| 90;72;90        | The first column is 90 points (1.25 inch); the second column is 72 points (1 inch); the third column is 90 points.  |
| 6 cm;0;6 cm     | The first column is 6 centimeters; the second column is hidden; the third column is 6 centimeters. Because part of the third column is visible, a horizontal scroll bar appears.          |
| 1.5 in;0;2.5 in | The first column is 1.5 inches, the second column is hidden, and the third column is 2.5 inches.  |
| 2 in;;2 in      | The first column is 2 inches, the second column is 1 inch (default), and the third column is 2 inches. Because only half of the third column is visible, a horizontal scroll bar appears. |
| (Blank)         | All three columns are the same width (1.33 inches).   |

## Remarks

In a combo box, the system displays the column designated by the **TextColumn** property in the text box portion of the control.

# Count Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proCountC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proCountX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proCountA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proCountS"}

Returns the number of objects in a collection.

## Syntax

*object*.Count

The **Count** property syntax has these parts:

| Part          | Description               |
|---------------|---------------------------|
| <i>object</i> | Required. A valid object. |

## Remarks

The **Count** property is read-only.

Note that the index value for the first page or tab of a collection is zero, the value for the second page or tab is one, and so on.

## CurLine Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proCurLineC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proCurLineX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proCurLineA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proCurLineS"}

Specifies the current line of a control.

### Syntax

*object*.**CurLine** [= *Long*]

The **CurLine** property syntax has these parts:

| Part          | Description  |
|---------------|--|
| <i>object</i> | Required. A valid object.                          |
| <i>Long</i>   | Optional. Specifies the current line of a control. |

### Remarks

The current line of a control is the line that contains the insertion point. The number of the first line is zero.

The **CurLine** property is valid when the control has the focus.

## CurTargetX Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proCurTargetXC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proCurTargetXX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proCurTargetXA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proCurTargetXS"}

Retrieves the preferred horizontal position of the insertion point in a multiline **TextBox** or **ComboBox**.

### Syntax

*object*.**CurTargetX** [ =*Long*]

The **CurTargetX** property syntax has these parts:

| Part          | Description   |
|---------------|---|
| <i>object</i> | Required. A valid object.   |
| <i>Long</i>   | Optional. Indicates the preferred position, measured in himetric units. A himetric is 0.0001 meter. |

### Remarks

The target position is relative to the left edge of the control. If the length of a line is less than the value of the **CurTargetX** property, you can place the insertion point at the end of the line. The value of **CurTargetX** changes when the user sets the insertion point or when the **CurX** property is set. **CurTargetX** is read-only.

The return value is valid when the object has focus.

You can use **CurTargetX** and **CurX** to move the insertion point as the user scrolls through the contents of a multiline **TextBox** or **ComboBox**. When the user moves the insertion point to another line of text by scrolling the content of the object, **CurTargetX** specifies the preferred position for the insertion point. **CurX** is set to this value if the line of text is longer than the value of **CurTargetX**. Otherwise, **CurX** is set to the end of the line of text.

## CurX Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proCurXC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proCurXX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proCurXA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proCurXS"}

Specifies the current horizontal position of the insertion point in a multiline **TextBox** or **ComboBox**.

### Syntax

*object*.**CurX** [= *Long*]

The **CurX** property syntax has these parts:

| Part          | Description  |
|---------------|--|
| <i>object</i> | Required. A valid object.  |
| <i>Long</i>   | Optional. Indicates the current position, measured in himetrics. A himetric is 0.0001 meter. |

### Remarks

The **CurX** property applies to a multiline **TextBox** or **ComboBox**. The return value is valid when the object has the focus.

You can use **CurTargetX** and **CurX** to position the insertion point as the user scrolls through the contents of a multiline **TextBox** or **ComboBox**. When the user moves the insertion point to another line of text by scrolling the content of the object, **CurTargetX** specifies the preferred position for the insertion point. **CurX** is set to this value if the line of text is longer than the value of **CurTargetX**. Otherwise, **CurX** is set to the end of the line of text.



## Delay Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proDelayC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proDelayX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proDelayA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proDelayS"}

Specifies the delay for the SpinUp, SpinDown, and Change events on a **SpinButton** or **ScrollBar**.

### Syntax

*object*.**Delay** [= *Long*]

The **Delay** property syntax has these parts:

| Part          | Description   |
|---------------|---|
| <i>object</i> | Required. A valid object.                             |
| <i>Long</i>   | Optional. The delay, in milliseconds, between events. |

### Remarks

The **Delay** property affects the amount of time between consecutive SpinUp, SpinDown, and Change events generated when the user clicks and holds down a button on a **SpinButton** or **ScrollBar**. The first event occurs immediately. The delay to the second occurrence of the event is five times the value of the specified **Delay**. This initial lag makes it easy to generate a single event rather than a stream of events.

After the initial lag, the interval between events is the value specified for **Delay**.

The default value of **Delay** is 50 milliseconds. This means the object initiates the first event after 250 milliseconds (5 times the specified value) and initiates each subsequent event after 50 milliseconds.

# DragBehavior Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proDragBehaviorC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proDragBehaviorX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proDragBehaviorA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proDragBehaviorS"}

Specifies whether the system enables the drag-and-drop feature for a **TextBox** or **ComboBox**.

## Syntax

*object*.**DragBehavior** [= *fmDragBehavior*]

The **DragBehavior** property syntax has these parts:

| Part                  | Description   |
|-----------------------|---|
| <i>object</i>         | Required. A valid object.   |
| <i>fmDragBehavior</i> | Optional. Specifies whether the drag-and-drop feature is enabled. |

## Settings

The settings for *fmDragBehavior* are:

| Constant                      | Value | Description                                      |
|-------------------------------|-------|--|
| <i>fmDragBehaviorDisabled</i> | 0     | Does not allow a drag-and-drop action (default). |
| <i>fmDragBehaviorEnabled</i>  | 1     | Allows a drag-and-drop action.                   |

## Remarks

If the **DragBehavior** property is enabled, dragging in a text box or combo box starts a drag-and-drop operation on the selected text. If **DragBehavior** is disabled, dragging in a text box or combo box selects text.

The drop-down portion of a **ComboBox** does not support drag-and-drop processes, nor does it support selection of list items within the text.

**DragBehavior** has no effect on a **ComboBox** whose **Style** property is set to **fmStyleDropDownList**.

**Note** You can combine the effects of the **EnterFieldBehavior** property and **DragBehavior** to create a large number of text box styles.

# DropButtonStyle Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proDropButtonStyleC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proDropButtonStyleX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proDropButtonStyleA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proDropButtonStyleS"}

Specifies the symbol displayed on the drop button in a **ComboBox**.

## Syntax

*object*.DropButtonStyle [= *fmDropButtonStyle*]

The **DropButtonStyle** property syntax has these parts:

| Part                     | Description                                  |
|--------------------------|--|
| <i>object</i>            | Required. A valid object.                    |
| <i>fmDropButtonStyle</i> | Optional. The appearance of the drop button. |

## Settings

The settings for *fmDropButtonStyle* are:

| Constant                         | Value | Description  |
|----------------------------------|-------|--|
| <i>fmDropButtonStylePlain</i>    | 0     | Displays a plain button, with no symbol.                 |
| <i>fmDropButtonStyleArrow</i>    | 1     | Displays a down arrow (default).                         |
| <i>fmDropButtonStyleEllipsis</i> | 2     | Displays an ellipsis (.).                                |
| <i>fmDropButtonStyleReduce</i>   | 3     | Displays a horizontal line like an underscore character. |

## Remarks

The recommended setting for showing items in a list is **fmDropButtonStyleArrow**. If you want to use the drop button in another way, such as to display a dialog box, specify **fmDropButtonStyleEllipsis**, **fmDropButtonStylePlain**, or **fmDropButtonStyleReduce** and trap the DropButtonClick event.

# Enabled Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proEnabledC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proEnabledX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proEnabledA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proEnabledS"}

Specifies whether a control can receive the focus and respond to user-generated events.

## Syntax

*object*.**Enabled** [= *Boolean*]

The **Enabled** property syntax has these parts:

| Part           | Description  |
|----------------|--|
| <i>object</i>  | Required. A valid object.  |
| <i>Boolean</i> | Optional. Specifies whether the object can respond to user-generated events. |

## Settings

The settings for *Boolean* are:

| Value        | Description  |
|--------------|--|
| <b>True</b>  | The control can receive the focus and respond to user-generated events, and is accessible through code (default).  |
| <b>False</b> | The user cannot interact with the control by using the mouse, keystrokes, accelerators, or hot keys. The control is generally still accessible through code. |

## Remarks

Use the **Enabled** property to enable and disable controls. A disabled control appears dimmed, while an enabled control does not. Also, if a control displays a bitmap, the bitmap is dimmed whenever the control is dimmed. If **Enabled** is **False** for an **Image**, the control does not initiate events but does not appear dimmed.

The **Enabled** and **Locked** properties work together to achieve the following effects:

- If **Enabled** and **Locked** are both **True**, the control can receive focus and appears normally (not dimmed) in the HTML Layout. The user can copy, but not edit, data in the control.
- If **Enabled** is **True** and **Locked** is **False**, the control can receive focus and appears normally in the HTML Layout. The user can copy and edit data in the control.
- If **Enabled** is **False** and **Locked** is **True**, the control cannot receive focus and is dimmed in the HTML Layout. The user can neither copy nor edit data in the control.
- If **Enabled** and **Locked** are both **False**, the control cannot receive focus and is dimmed in the HTML Layout. The user can neither copy nor edit data in the control.

You can combine the settings of the **Enabled** and the **TabStop** properties to prevent the user from selecting a command button with **TAB**, while still allowing the user to click the button. Setting **TabStop** to **False** means that the command button won't appear in the tab order. However, if **Enabled** is **True**, then the user can still click the command button, as long as **TakeFocusOnClick** is set to **True**.

When the user tabs into an enabled **TabStrip**, the first page or tab in the control receives the focus. If the first page or tab of a **TabStrip** is disabled, the first enabled page or tab of that control receives the focus. If all pages or tabs of a or **TabStrip** are disabled, the control is disabled and cannot receive the focus.

# EnterFieldBehavior Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proEnterFieldBehaviorC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proEnterFieldBehaviorX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proEnterFieldBehaviorA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proEnterFieldBehaviorS"}

Specifies the selection behavior when entering a **TextBox** or **ComboBox**.

## Syntax

*object*.**EnterFieldBehavior** [= *fmEnterFieldBehavior*]

The **EnterFieldBehavior** property syntax has these parts:

| Part                        | Description                               |
|-----------------------------|---|
| <i>object</i>               | Required. A valid object.                 |
| <i>fmEnterFieldBehavior</i> | Optional. The desired selection behavior. |

## Settings

The settings for *fmEnterFieldBehavior* are:

| Constant                                   | Value | Description  |
|--|-------|--|
| <i>fmEnterFieldBehaviorSelectAll</i>       | 0     | Selects the entire contents of the edit region when entering the control (default).  |
| <i>fmEnterFieldBehaviorRecallSelection</i> | 1     | Leaves the selection unchanged. Visually, this uses the selection that was in effect the last time the control was active. |

## Remarks

The **EnterFieldBehavior** property controls the way text is selected when the user tabs to the control, not when the control receives focus as a result of the **SetFocus** method. Following **SetFocus**, the contents of the control are not selected and the insertion point appears after the last character in the control's edit region.

# EnterKeyBehavior Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proEnterKeyBehaviorC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proEnterKeyBehaviorX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proEnterKeyBehaviorA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proEnterKeyBehaviorS"}

Defines the effect of pressing ENTER in a **TextBox**.

## Syntax

*object*.**EnterKeyBehavior** [= *Boolean*]

The **EnterKeyBehavior** property syntax has these parts:

| Part           | Description                                       |
|----------------|---|
| <i>object</i>  | Required. A valid object.                         |
| <i>Boolean</i> | Optional. Specifies the effect of pressing ENTER. |

## Settings

The settings for *Boolean* are:

| Value        | Description   |
|--------------|---|
| <b>True</b>  | Pressing ENTER creates a new line.  |
| <b>False</b> | Pressing ENTER moves the focus to the next object in the tab order (default). |

## Remarks

The **EnterKeyBehavior** and **MultiLine** properties are closely related. The values described above only apply if **MultiLine** is **True**. If **MultiLine** is **False**, pressing ENTER always moves the focus to the next control in the tab order regardless of the value of **EnterKeyBehavior**.

The effect of pressing CTRL+ENTER also depends on the value of **MultiLine**. If **MultiLine** is **True**, pressing CTRL+ENTER creates a new line regardless of the value of **EnterKeyBehavior**. If **MultiLine** is **False**, pressing CTRL+ENTER has no effect.

# ForeColor Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proForeColorC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proForeColorX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proForeColorA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proForeColorS"}

Specifies the foreground color of an object.

## Syntax

*object*.ForeColor [= Long]

The **ForeColor** property syntax has these parts:

| Part          | Description  |
|---------------|--|
| <i>object</i> | Required. A valid object.  |
| <i>Long</i>   | Optional. A value or constant that determines the foreground color of an object. |

## Settings

You can use any integer that represents a valid color. You can also specify a color by using the RGB function with red, green, and blue color components. The value of each color component is an integer that ranges from zero to 255. For example, you can specify teal blue as the integer value 4966415 or as red, green, and blue color components 15, 200, 75.

## Remarks

Use the **ForeColor** property for controls on HTML Layouts to make them easy to read or to convey a special meaning. For example, if a text box reports the number of units in stock, you can change the color of the text when the value falls below the reorder level.

For a **ScrollBar** or **SpinButton**, **ForeColor** sets the color of the arrows. For a **Font** object, **ForeColor** determines the color of the text.

## GroupName Property

```
{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proGroupNameC"}           {ewc  
HLP95EN.DLL,DYNALINK,"Example":"f3proGroupNameX":1}           {ewc HLP95EN.DLL,DYNALINK,"Applies  
To":"f3proGroupNameA"}           {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proGroupNameS"}
```

Creates a group of mutually exclusive **OptionButton** controls.

### Syntax

*object*.**GroupName** [= *String*]

The **GroupName** syntax has these parts:

| <b>Part</b>   | <b>Description</b>   |
|---------------|--|
| <i>object</i> | Required. A valid <b>OptionButton</b> .  |
| <i>String</i> | Optional. The name of the group that includes the <b>OptionButton</b> . Use the same setting for all buttons in the group. The default setting is an empty string. |

### Remarks

You can create buttons with transparent backgrounds, which can improve the visual appearance of your HTML Layout.

Clicking one button in a group sets all other buttons in the same group to **False**. All option buttons with the same **GroupName** within a single container are mutually exclusive. You can use the same group name in two containers, but doing so creates two groups (one in each container) rather than one group that includes both containers.



# Height, Width Properties

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proHeightC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proHeightX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proHeightA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proHeightS"}

The height or width, in points, of an object.

## Syntax

*object*.**Height** [= *Single*]

*object*.**Width** [= *Single*]

The **Height** and **Width** property syntaxes have these parts:

| Part          | Description  |
|---------------|--|
| <i>object</i> | Required. A valid object.  |
| <i>Single</i> | Optional. A numeric expression specifying the dimensions of an object. |

## Remarks

The **Height** and **Width** properties are automatically updated when you move or size a control. If you change the size of a control, the **Height** or **Width** property stores the new height or width. If you specify a setting for the **Left** or **Top** property that is less than zero, that value will be used to calculate the height or width of the control, but a portion of the control will not be visible on the HTML Layout.

If you move a control from one part of an HTML Layout to another, the setting of **Height** or **Width** changes only if you size the control as you move it. The settings of the control's **Left** and **Top** properties will change to reflect the control's new position relative to the edges of the HTML Layout that contains it.

The value assigned to **Height** or **Width** must be greater than or equal to zero. For most systems, the recommended range of values is from 0 to +32,767. Higher values may also work depending on your system configuration.

## HideSelection Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proHideSelectionC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proHideSelectionX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proHideSelectionA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proHideSelectionS"}

Specifies whether selected text remains highlighted when a control does not have the focus.

### Syntax

*object*.**HideSelection** [= *Boolean*]

The **HideSelection** property syntax has these parts:

| Part           | Description  |
|----------------|--|
| <i>object</i>  | Required. A valid object.  |
| <i>Boolean</i> | Optional. Specifies whether the selected text remains highlighted even when the control does not have the focus. |

### Settings

The settings for *Boolean* are:

| Value        | Description  |
|--------------|--|
| <b>True</b>  | Selected text is not highlighted unless the control has the focus (default). |
| <b>False</b> | Selected text always appears highlighted.                                    |

### Remarks

You can use the **HideSelection** property to maintain highlighted text when another HTML Layout or a dialog box receives the focus, such as in a spell-checking procedure.

# IMEMode Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proIMEModeC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proIMEModeX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proIMEModeA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proIMEModeS"}

Specifies the default run time mode of the Input Method Editor (IME) for a control. This property applies only to applications written for the Far East and is ignored in other applications.

## Syntax

*object*.IMEMode [= *fmIMEMode*]

The **IMEMode** property syntax has these parts:

| Part             | Description  |
|------------------|--|
| <i>object</i>    | Required. A valid object.                            |
| <i>fmIMEMode</i> | Optional. The mode of the Input Method Editor (IME). |

## Settings

The settings for *fmIMEMode* are:

| Constant                    | Value | Description                                  |
|-----------------------------|-------|--|
| <i>fmIMEModeNoOp</i>        | 0     | Does not control IME (default).              |
| <i>fmIMEModeOn</i>          | 1     | IME on.                                      |
| <i>fmIMEModeOff</i>         | 2     | IME off. English mode.                       |
| <i>fmIMEModeDisable</i>     | 3     | IME off. User can't turn on IME by keyboard. |
| <i>fmIMEModeHiragana</i>    | 4     | IME on with Full-width Hiragana mode.        |
| <i>fmIMEModeKatakanaDbl</i> | 5     | IME on with Full-width Katakana mode.        |
| <i>fmIMEModeKatakanaSng</i> | 6     | IME on with Half-width Katakana mode.        |
| <i>fmIMEModeAlphaDbl</i>    | 7     | IME on with Full-width Alphanumeric mode.    |
| <i>fmIMEModeAlphaSng</i>    | 8     | IME on with Half-width Alphanumeric mode.    |

The **fmIMEModeNoOp** setting indicates that the mode of the IME does not change when the control receives focus at run time. For any other value, the mode of the IME is set to the value specified by the **IMEMode** property when the control receives focus at run time.

## Remarks

There are two ways to set the mode of the IME. One is through the IME toolbar. The other is with a control's **IMEMode** property, which sets or returns the current mode of the IME. This property allows dynamic control of the IME through code.

The following example explains how **IMEMode** interacts with the IME toolbar. Assume that you have designed an HTML Layout with `TextBox1` and `CheckBox1`. You have set `TextBox1.IMEMode` to 0, and you have set `CheckBox1.IMEMode` to 1. While in design mode you have used the IME toolbar to put the IME in mode 2.

When you run the HTML Layout, the IME begins in mode 2. If you click `TextBox1`, the IME mode does not change because **IMEMode** for this control is 0. If you click `CheckBox1`, the IME changes to mode 1, because **IMEMode** for this control is 1. If you click again on `TextBox1`, the IME remains in mode 1

(**IMEMode** is 0, so the IME retains its last setting).

However, you can override **IMEMode**. For example, assume you click **CheckBox1** and the IME enters mode 1, as defined by **IMEMode** for the **CheckBox**. If you then use the IME toolbar to put the IME in mode 3, then the IME will be set to mode 3 when you click the control. This does not change the value of the property, it overrides the property until the next time you run the HTML Layout.

# Index Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proIndexC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proIndexX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proIndexA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proIndexS"}

The position of a **Tab** object within a **Tabs** collection.

## Syntax

*object*.**Index** [= *Integer*]

The **Index** property syntax has these parts:

| Part           | Description  |
|----------------|--|
| <i>object</i>  | Required. A valid object.  |
| <i>Integer</i> | Optional. The index of the currently selected <b>Tab</b> object. |

## Remarks

The **Index** property specifies the order in which tabs appear. Changing the value of **Index** visually changes the order of **Tabs** on a **TabStrip**. The index value for the first page or tab is zero, the index value of the second page or tab is one, and so on.

In a **TabStrip**, **Index** refers to the tab only.

## InsideHeight, InsideWidth Properties

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proInsideHeightC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proInsideHeightX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proInsideHeightA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proInsideHeightS"}

**InsideHeight** returns the height, in points, of the client region inside an HTML Layout. **InsideWidth** returns the width, in points, of the client region inside an HTML Layout.

### Syntax

*object*.**InsideHeight** [=Single]

*object*.**InsideWidth** [=Single]

The **InsideHeight** and **InsideWidth** property syntaxes have these parts:

| Part          | Description   |
|---------------|---|
| <i>object</i> | Required. A valid object.                           |
| <i>Single</i> | Optional. The height or width of the client region. |

### Remarks

If the region includes a scroll bar, the returned value does not include the height or width of the scroll bar.

# IntegralHeight Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proIntegralHeightC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proIntegralHeightX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proIntegralHeightA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proIntegralHeightS"}

Indicates whether a **ListBox** or **TextBox** displays full lines of text in a list or partial lines.

## Syntax

*object*.**IntegralHeight** [= *Boolean*]

The **IntegralHeight** property syntax has these parts:

| Part           | Description  |
|----------------|--|
| <i>object</i>  | Required. A valid object.  |
| <i>Boolean</i> | Optional. Specifies whether the list displays partial lines of text. |

## Settings

The settings for *Boolean* are:

| Value        | Description   |
|--------------|---|
| <b>True</b>  | The list resizes itself to display only complete items (default).                   |
| <b>False</b> | The list does not resize itself even if the item is too tall to display completely. |

## Remarks

The **IntegralHeight** property relates to the height of the list, just as the **AutoSize** property relates to the width of the list.

If **IntegralHeight** is **True**, the list box automatically resizes when necessary to show full rows. If **False**, the list remains a fixed size; if items are taller than the available space in the list, the entire item is not shown.

## LargeChange Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proLargeChangeC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proLargeChangeX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proLargeChangeA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proLargeChangeS"}

Specifies the amount of movement that occurs when the user clicks between the scroll box and scroll arrow.

### Syntax

*object*.**LargeChange** [= *Long*]

The **LargeChange** property syntax has these parts:

| Part          | Description  |
|---------------|--|
| <i>object</i> | Required. A valid object.  |
| <i>Long</i>   | Optional. An integer that specifies the amount of change to the <b>Value</b> property. |

### Remarks

The **LargeChange** property applies only to the **ScrollBar**. It does not apply to the scrollbars in other controls such as a **TextBox** or a drop-down **ComboBox**.

The value of **LargeChange** is the amount by which the **ScrollBar's Value** property changes when the user clicks the area between the scroll box and scroll arrow. The direction of the movement is always toward the place where the user clicks. For example, in a horizontal **ScrollBar**, clicking to the left of the scroll box moves the scroll box to the left. In a vertical **ScrollBar**, clicking above the scroll box moves the scroll box up.

**LargeChange** does not have units. Any integer is a valid setting for **LargeChange**. The recommended range of values is from -32,767 to +32,767, and the value must be between the values of the **Max** and **Min** properties of the **ScrollBar**.



## Left, Top Properties

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proLeftC"}  
{ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proLeftA"}

{ewc HLP95EN.DLL, DYNALINK, "Example":"f3proLeftX":1}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proLeftS"}

The distance between a control and the left or top edge of the HTML Layout that contains it.

### Syntax

*object*.**Left** [= *Single*]

*object*.**Top** [= *Single*]

The **Left** and **Top** property syntaxes have these parts:

| Part          | Description   |
|---------------|---|
| <i>object</i> | Required. A valid object.   |
| <i>Single</i> | Optional. A numeric expression specifying the coordinates of an object. |

### Settings

Setting the **Left** or **Top** property to 0 places the control's edge at the left or top edge of its container.

### Remarks

For most systems, the recommended range of values for **Left** and **Top** is from -32,767 to +32,767. Other values may also work depending on your system configuration. For a **ComboBox**, values of **Left** and **Top** apply to the text portion of the control, not to the list portion. When you move or size a control, its new **Left** setting is automatically entered in the Properties window. When you print an HTML Layout, the control's horizontal or vertical location is determined by its **Left** or **Top** setting.

## LineCount Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proLineCountC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proLineCountX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proLineCountA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proLineCounts"}

Returns the number of text lines in a **TextBox** or **ComboBox**.

### Syntax

*object*.**LineCount** [ =*Long*]

The **LineCount** property syntax has these parts:

| Part          | Description  |
|---------------|--|
| <i>object</i> | Required. A valid object.  |
| <i>Long</i>   | Return value. Specifies the number of text lines in the control. |

### Remarks

The **LineCount** property is read-only. The returned value is from 0 to one less than the value of **LineCount**.

**Note** A **ComboBox** will have only one line.

## List Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proListC"}  
{ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proListA"}

{ewc HLP95EN.DLL, DYNALINK, "Example": "f3proListX": 1}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proListS"}

Returns or sets the list entries of a **ListBox** or **ComboBox**.

### Syntax

*object*.**List**( *row*, *column* ) [= *Variant*]

The **List** property syntax has these parts:

| Part           | Description  |
|----------------|--|
| <i>object</i>  | Required. A valid object.  |
| <i>row</i>     | Required. An integer with a range from 0 to one less than the number of entries in the list. |
| <i>column</i>  | Required. An integer with a range from 0 to one less than the number of columns.             |
| <i>Variant</i> | Optional. The contents of the specified entry in the <b>ListBox</b> or <b>ComboBox</b> .     |

### Settings

Row and column numbering begins with zero. That is, the row number of the first row in the list is zero; the column number of the first column is zero. The number of the second row or column is 1, and so on.

### Remarks

The **List** property works with the **ListCount** and **ListIndex** properties. Use **List** in code to access list items. A list is a variant array; each item in the list has a row number and a column number.

Initially, **ComboBox** and **ListBox** contain an empty list.

**Note** To specify items you want to display in a **ComboBox** or **ListBox**, use the **AddItem** method. To remove items, use the **RemoveItem** method.

You can also use **List** to copy an entire two-dimensional array of values to a control. This lets you quickly load a list of choices rather than using **AddItem** to individually load each element of the list.

## ListCount Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proListCountC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proListCountX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proListCountA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proListCountS"}

Returns the number of list entries in a control.

### Syntax

*Long*=*object*.**ListCount**

The **ListCount** property syntax has these parts:

| Part          | Description  |
|---------------|--|
| <i>object</i> | Required. A valid object.  |
| <i>Long</i>   | Return value. Reports the number of entries in a control. The default value is zero (0). |

### Remarks

The **ListCount** property is read-only. **ListCount** is the number of rows over which you can scroll. **ListRows** is the maximum to display at once. **ListCount** is always one greater than the largest value for the **ListIndex** property, because index numbers begin with 0 and the count of items begins with 1. If no item is selected, **ListCount** is 0 and **ListIndex** is -1.

# ListIndex Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proListIndexC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proListIndexX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proListIndexA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proListIndexS"}

Identifies the currently selected item in a **ListBox** or **ComboBox**.

## Syntax

*object*.ListIndex [= *Variant*]

The **ListIndex** property syntax has these parts:

| Part           | Description   |
|----------------|---|
| <i>object</i>  | Required. A valid object.                             |
| <i>Variant</i> | Optional. The currently selected item in the control. |

## Remarks

The **ListIndex** property contains an index of the selected row in a list. Values of **ListIndex** range from -1 to one less than the total number of rows in a list (that is, **ListCount** - 1). When no rows are selected, **ListIndex** returns -1. When the user selects a row in a **ListBox** or **ComboBox**, the system sets the **ListIndex** value. The **ListIndex** value of the first row in a list is 0, the value of the second row is 1, and so on.

**Note** If you use the **MultiSelect** property to create a **ListBox** that allows multiple selections, the **Selected** property of the **ListBox** (rather than the **ListIndex** property) identifies the selected rows. The **Selected** property is an array with the same number of values as the number of rows in the **ListBox**. For each row in the list box, **Selected** is **True** if the row is selected and **False** if it is not. In a **ListBox** that allows multiple selections, **ListIndex** returns the index of the row that has focus, regardless of whether that row is currently selected.

## ListRows Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proListRowsC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proListRowsX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proListRowsA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proListRowsS"}

Specifies the maximum number of rows to display in the list before displaying a vertical scroll bar.

### Syntax

*object*.**ListRows** [= *Long*]

The **ListRows** property syntax has these parts:

| <b>Part</b>   | <b>Description</b>  |
|---------------|---|
| <i>object</i> | Required. A valid object.   |
| <i>Long</i>   | Optional. An integer indicating the maximum number of rows. The default value is 8. |

### Remarks

If the number of items in the list exceeds the value of the **ListRows** property, a scroll bar appears at the right edge of the list box portion of the combo box.

# ListStyle Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proListStyleC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proListStyleX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proListStyleA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proListStyleS"}

Specifies the visual appearance of the list in a **ListBox** or **ComboBox**.

## Syntax

*object*.**ListStyle** [= *fmListStyle*]

The **ListStyle** property syntax has these parts:

| Part               | Description                             |
|--------------------|---|
| <i>object</i>      | Required. A valid object.               |
| <i>fmListStyle</i> | Optional. The visual style of the list. |

## Settings

The settings for *fmListStyle* are:

| Constant                 | Value | Description  |
|--------------------------|-------|--|
| <i>fmListStylePlain</i>  | 0     | Looks like a regular list box, with the background of items highlighted.   |
| <i>fmListStyleOption</i> | 1     | Shows option buttons, or check boxes for a multiselect list (default). When the user selects an item from the group, the option button associated with that item is selected and the option buttons for the other items in the group are deselected. |

## Remarks

The **ListStyle** property lets you change the visual presentation of a **ListBox** or **ComboBox**. By specifying a setting other than **fmListStylePlain**, you can present the contents of either control as a group of individual items, with each item including a visual cue to indicate whether it is selected.

If the control supports a single selection (the **MultiSelect** property is set to **fmMultiSelectSingle**), the user can press one button in the group. If the control supports multiselect, the user can press two or more buttons in the group.

## ListWidth Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proListWidthC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proListWidthX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proListWidthA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proListWidthS"}

Specifies the width of the list in a **ComboBox**.

### Syntax

*object*.ListWidth [= *Variant*]

The **ListWidth** property syntax has these parts:

| Part           | Description  |
|----------------|--|
| <i>object</i>  | Required. A valid object.  |
| <i>Variant</i> | Optional. The width of the list. A value of zero makes the list as wide as the <b>ComboBox</b> . The default value is to make the list as wide as the text portion of the control. |

### Remarks

If you want to display a multicolumn list, enter a value that will make the list box wide enough to fit all the columns.

**Tip** When designing combo boxes, be sure to leave enough space to display your data and for a vertical scroll bar.



# Locked Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proLockedC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proLockedX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proLockedA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proLockedS"}

Specifies whether a control can be edited.

## Syntax

*object*.**Locked** [= *Boolean*]

The **Locked** property syntax has these parts:

| Part           | Description  |
|----------------|--|
| <i>object</i>  | Required. A valid object.                              |
| <i>Boolean</i> | Optional. Specifies whether the control can be edited. |

## Settings

The settings for *Boolean* are:

| Value        | Description                       |
|--------------|-----------------------------------|
| <b>True</b>  | You can't edit the value.         |
| <b>False</b> | You can edit the value (default). |

## Remarks

When a control is locked and enabled, it can still initiate events and can still receive the focus.

# MatchEntry Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proMatchEntryC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proMatchEntryX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proMatchEntryA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proMatchEntryS"}

Returns or sets a value indicating how a **ListBox** or **ComboBox** searches its list as the user types.

## Syntax

*object.MatchEntry* [= *fmMatchEntry*]

The **MatchEntry** property syntax has these parts:

| Part                | Description   |
|---------------------|---|
| <i>object</i>       | Required. A valid object.                             |
| <i>fmMatchEntry</i> | Optional. The rule used to match entries in the list. |

## Settings

The settings for *fmMatchEntry* are:

| Constant                       | Value | Description   |
|--------------------------------|-------|---|
| <i>fmMatchEntryFirstLetter</i> | 0     | Basic matching. The control searches for the next entry that starts with the character entered. Repeatedly typing the same letter <u>cycles</u> through all entries beginning with that letter. |
| <i>FmMatchEntryComplete</i>    | 1     | Extended matching. As each character is typed, the control searches for an entry matching all characters entered (default).   |
| <i>FmMatchEntryNone</i>        | 2     | No matching.  |

## Remarks

The **MatchEntry** property searches entries from the **TextColumn** property of a **ListBox** or **ComboBox**.

The control searches the column identified by **TextColumn** for an entry that matches the user's typed entry. Upon finding a match, the row containing the match is selected, the contents of the column are displayed, and the contents of its **BoundColumn** property become the value of the control. If the match is unambiguous, finding the match initiates the Click event.

The control initiates the Click event as soon as the user types a sequence of characters that match exactly one entry in the list. As the user types, the entry is compared with the current row in the list and with the next row in the list. When the entry matches only the current row, the match is unambiguous.

In ActiveX Control Pad, this is true regardless of whether the list is sorted. This means the control finds the first occurrence that matches the entry, based on the order of items in the list. For example, entering either "abc" or "bc" will initiate the Click event for the following list:

```
abcde  
bcdef  
abcxyz  
bchij
```

Note that in either case, the matched entry is not unique; however, it is sufficiently different from the adjacent entry that the control interprets the match as unambiguous and initiates the Click event.

## MatchFound Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proMatchFoundC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proMatchFoundX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proMatchFoundA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proMatchFoundS"}

Indicates whether the text that a user has typed into a combo box matches any of the entries in the list.

### Syntax

*object*.**MatchFound** [ =*Boolean*]

The **MatchFound** property syntax has these parts:

| Part           | Description  |
|----------------|--|
| <i>object</i>  | Required. A valid object.                                |
| <i>Boolean</i> | Optional. Specifies whether a matching record was found. |

### Return Values

The **MatchFound** property return values are:

| Value        | Description   |
|--------------|---|
| <b>True</b>  | The contents of the <b>Value</b> property matches one of the records in the list.     |
| <b>False</b> | The contents of <b>Value</b> does not match any of the records in the list (default). |

### Remarks

The **MatchFound** property is read-only. It is not applicable when the **MatchEntry** property is set to **fmMatchEntryNone**.

## MatchRequired Property

```
{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proMatchRequiredC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proMatchRequiredX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proMatchRequiredA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proMatchRequiredS"}
```

Specifies whether a value entered in the text portion of a **ComboBox** must match an entry in the existing list portion of the control. The user can enter non-matching values, but may not leave the control until a matching value is entered.

### Syntax

*object*.**MatchRequired** [= *Boolean*]

The **MatchRequired** property syntax has these parts:

| Part           | Description   |
|----------------|---|
| <i>object</i>  | Required. A valid object.   |
| <i>Boolean</i> | Optional. Specifies whether the text entered must match an existing item in the list. |

### Settings

The settings for *Boolean* are:

| Value        | Description   |
|--------------|---|
| <b>True</b>  | The text entered must match an existing list entry.                         |
| <b>False</b> | The text entered can be different from all existing list entries (default). |

### Remarks

If the **MatchRequired** property is **True**, the user cannot exit the **ComboBox** until the text entered matches an entry in the existing list. **MatchRequired** maintains the integrity of the list by requiring the user to select an existing entry.

**Note** Not all containers enforce this property.

## Max, Min Properties

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proMaxC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proMaxX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proMaxA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proMaxS"}

Specify the maximum and minimum acceptable values for the **Value** property of a **ScrollBar** or **SpinButton**.

### Syntax

*object*.**Max** [= *Long*]

*object*.**Min** [= *Long*]

The **Max** and **Min** property syntaxes have these parts:

| Part          | Description   |
|---------------|---|
| <i>object</i> | Required. A valid object.   |
| <i>Long</i>   | Optional. A numeric expression specifying the maximum or minimum <b>Value</b> property setting. |

### Remarks

Clicking a **SpinButton** or moving the scroll box in a **ScrollBar** changes the **Value** property of the control.

The value for the **Max** property corresponds to the lowest position of a vertical **ScrollBar** or the rightmost position of a horizontal **ScrollBar**. The value for the **Min** property corresponds to the highest position of a vertical **ScrollBar** or the leftmost position of a horizontal **ScrollBar**.

Any integer is an acceptable setting for this property. The recommended range of values is from –32,767 to +32,767. The default value is 1.

**Note** **Min** and **Max** refer to locations, not to relative values, on the **ScrollBar**. That is, the value of **Max** could be less than the value of **Min**. If this is the case, moving toward the **Max** (bottom) position means decreasing **Value**; moving toward the **Min** (top) position means increasing **Value**.

# MaxLength Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proMaxLengthC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proMaxLengthX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proMaxLengthA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proMaxLengthS"}

Specifies the maximum number of characters a user can enter in a **TextBox** or **ComboBox**.

## Syntax

*object*.**MaxLength** [= *Long*]

The **MaxLength** property syntax has these parts:

| Part          | Description   |
|---------------|---|
| <i>object</i> | Required. A valid object.   |
| <i>Long</i>   | Optional. An integer indicating the allowable number of characters. |

## Remarks

Setting the **MaxLength** property to 0 indicates there is no limit other than that created by memory constraints.

# Mouselcon Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proMouselconC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proMouselconX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proMouselconA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proMouselconS"}

Assigns a custom icon to an object.

## Syntax

*object*.**Mouselcon** = **LoadPicture**( *pathname* )

The **Mouselcon** property syntax has these parts:

| Part            | Description  |
|-----------------|--|
| <i>object</i>   | Required. A valid object.  |
| <i>pathname</i> | Required. A string expression specifying the path and filename of the file containing the custom icon. |

## Remarks

The **Mouselcon** property is valid when the **MousePointer** property is set to 99. The mouse icon of an object is the image that appears when the user moves the mouse across that object.

To assign an image for the mouse pointer, you can either assign a picture to the **Mouselcon** property or load a picture from a file using the **LoadPicture** method.

## MousePointer Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proMousePointerC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proMousePointerX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proMousePointerA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proMousePointerS"}

Specifies the type of pointer displayed when the user positions the mouse over a particular object.

### Syntax

*object*.**MousePointer** [= *fmMousePointer*]

The **MousePointer** property syntax has these parts:

| Part                  | Description   |
|-----------------------|---|
| <i>object</i>         | Required. A valid object.                           |
| <i>fmMousePointer</i> | Optional. The shape you want for the mouse pointer. |

### Settings

The settings for *fmMousePointer* are:

| Constant                         | Value | Description  |
|----------------------------------|-------|--|
| <i>fmMousePointerDefault</i>     | 0     | Standard pointer. The image is determined by the object (default).   |
| <i>fmMousePointerArrow</i>       | 1     | Arrow.   |
| <i>fmMousePointerCross</i>       | 2     | Cross-hair pointer.  |
| <i>fmMousePointerIBeam</i>       | 3     | I-beam.  |
| <i>fmMousePointerSizeNESW</i>    | 6     | Double arrow pointing northeast and southwest.   |
| <i>fmMousePointerSizeNS</i>      | 7     | Double arrow pointing north and south.   |
| <i>fmMousePointerSizeNWSE</i>    | 8     | Double arrow pointing northwest and southeast.   |
| <i>fmMousePointerSizeWE</i>      | 9     | Double arrow pointing west and east.   |
| <i>fmMousePointerUpArrow</i>     | 10    | Up arrow.  |
| <i>fmMousePointerHourglass</i>   | 11    | Hourglass.   |
| <i>fmMousePointerNoDrop</i>      | 12    | "Not" symbol (circle with a diagonal line) on top of the object being dragged. Indicates an invalid drop target. |
| <i>fmMousePointerAppStarting</i> | 13    | Arrow with an hourglass.   |
| <i>fmMousePointerHelp</i>        | 14    | Arrow with a question mark.  |
| <i>fmMousePointerSizeAll</i>     | 15    | Size all cursor (arrows pointing north, south, east, and west).  |
| <i>fmMousePointerCustom</i>      | 99    | Uses the icon specified by the <b>MouseIcon</b> property.  |

### Remarks

Use the **MousePointer** property when you want to indicate changes in functionality as the mouse pointer passes over controls on an HTML Layout. For example, the hourglass setting (11) is useful to indicate that the user must wait for a process or operation to finish.



Some icons vary depending on system settings, such as the icons associated with desktop themes.

# MultiLine Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proMultiLineC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proMultiLineX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proMultiLineA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proMultiLineS"}

Specifies whether a control can accept and display multiple lines of text.

## Syntax

*object*.MultiLine [= Boolean]

The **MultiLine** property syntax has these parts:

| Part           | Description  |
|----------------|--|
| <i>object</i>  | Required. A valid object.  |
| <i>Boolean</i> | Optional. Specifies whether the control supports more than one line of text. |

## Settings

The settings for *Boolean* are:

| Value        | Description  |
|--------------|--|
| <b>True</b>  | The text is displayed across multiple lines (default). |
| <b>False</b> | The text is not displayed across multiple lines.       |

## Remarks

A multiline **TextBox** allows absolute line breaks and adjusts its quantity of lines to accommodate the amount of text it holds. If needed, a multiline control can have vertical scroll bars.

A single-line **TextBox** doesn't allow absolute line breaks and doesn't use vertical scroll bars.

Single-line controls ignore the value of the **WordWrap** property.

**Note** If you change **MultiLine** to **False** in a multiline **TextBox**, all the characters in the **TextBox** will be combined into one line. Non-printing characters will be displayed between lines of text.

## MultiRow Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proMultiRowC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proMultiRowX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proMultiRowA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proMultiRowS"}

Specifies whether the control has more than one row of tabs.

### Syntax

*object*.**MultiRow** [= *Boolean*]

The **MultiRow** property syntax has these parts:

| Part           | Description  |
|----------------|--|
| <i>object</i>  | Required. A valid object.  |
| <i>Boolean</i> | Optional. Specifies whether the control has more than one row of tabs. |

### Settings

The settings for *Boolean* are:

| Value        | Description                               |
|--------------|---|
| <b>True</b>  | Allows more than one row of tabs.         |
| <b>False</b> | Restricts tabs to a single row (default). |

### Remarks

The width and number of tabs determines the number of rows. Changing the control's size also changes the number of rows. This allows the developer to resize the control and ensure that tabs wrap to fit the control. If the **MultiRow** property is **False**, then truncation occurs if the width of the tabs exceeds the width of the control.

If **MultiRow** is **False** and tabs are truncated, there will be a small scroll bar on the **TabStrip** to allow scrolling to the other tabs or pages.

# MultiSelect Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proMultiSelectC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proMultiSelectX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proMultiSelectA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proMultiSelectS"}

Indicates whether the object permits multiple selections.

## Syntax

*object*.MultiSelect [= *fmMultiSelect*]

The **MultiSelect** property syntax has these parts:

| Part                 | Description   |
|----------------------|---|
| <i>object</i>        | Required. A valid object.                           |
| <i>fmMultiSelect</i> | Optional. The selection mode that the control uses. |

## Settings

The settings for *fmMultiSelect* are:

| Constant                     | Value | Description   |
|------------------------------|-------|---|
| <i>fmMultiSelectSingle</i>   | 0     | Only one item can be selected (default).  |
| <i>fmMultiSelectSimple</i>   | 1     | Pressing the SPACEBAR or clicking selects or deselects an item in the list.   |
| <i>fmMultiSelectExtended</i> | 2     | Pressing SHIFT and clicking the mouse, or pressing SHIFT and one of the arrow keys, extends the selection from the previously selected item to the current item. Pressing CTRL and clicking the mouse selects or deselects an item. |

## Remarks

When the **MultiSelect** property is set to *Extended* or *Simple*, you must use the list box's **Selected** property to determine the selected items. Also, the **Value** property of the control is always **Null**.

The **ListIndex** property returns the index of the row with the keyboard focus.

# ID Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proNameC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proNameX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proNameA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proNameS"}

Specifies the name of a control or an object, or the name of a font to associate with a **Font** object.

## Syntax

For Font

*Font.ID* [= *String*]

For all other controls and objects

*object.ID* [= *String*]

The **ID** property syntax has these parts:

| Part          | Description   |
|---------------|---|
| <i>object</i> | Required. A valid object.                                     |
| <i>String</i> | Optional. The name you want to assign to the font or control. |

## Settings

Guidelines for assigning a string to **ID**, such as the maximum length of the name, vary from one application to another.

## Remarks

For objects, the default value of **ID** consists of the object's class name followed by an integer. For example, the default name for the first **TextBox** you place on an HTML Layout is TextBox1. The default name for the second **TextBox** is TextBox2.

You can set the **ID** property for a control from the control's Properties window or, for controls added at run time, by using program statements. If you add a control at design time, you cannot modify its **ID** property at run time.

Each control added to an HTML Layout at design time must have a unique name.

For **Font** objects, **ID** identifies a particular typeface to use in the text portion of a control, object, or HTML Layout. The font's appearance on screen and in print may differ, depending on your computer and printer. If you select a font that your system can't display or that isn't installed, Windows substitutes a similar font.

# Orientation Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proOrientationC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proOrientationX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proOrientationA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proOrientationS"}

Specifies whether the **SpinButton** or **ScrollBar** is oriented vertically or horizontally.

## Syntax

*object*.Orientation [= *fmOrientation*]

The **Orientation** property syntax has these parts:

| Part                 | Description                           |
|----------------------|---------------------------------------|
| <i>object</i>        | Required. A valid object.             |
| <i>fmOrientation</i> | Optional. Orientation of the control. |

## Settings

The settings for *fmOrientation* are:

| Constant                       | Value | Description  |
|--------------------------------|-------|--|
| <i>fmOrientationAuto</i>       | -1    | Automatically determines the orientation based upon the dimensions of the control (default). |
| <i>fmOrientationVertical</i>   | 0     | Control is rendered vertically.  |
| <i>fmOrientationHorizontal</i> | 1     | Control is rendered horizontally.  |

## Remarks

If you specify automatic orientation, the height and width of the control determine whether the **SpinButton** or **ScrollBar** appears horizontally or vertically. For example, if the control is wider than it is tall, the **SpinButton** or **ScrollBar** appears horizontally; if the control is taller than it is wide, the **SpinButton** or **ScrollBar** appears vertically.

## PasswordChar Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proPasswordCharC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proPasswordCharX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proPasswordCharA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proPasswordCharS"}

Specifies whether placeholder characters are displayed instead of the characters actually entered in a **TextBox**.

### Syntax

*object*.**PasswordChar** [= *String*]

The **PasswordChar** property syntax has these parts:

| Part          | Description   |
|---------------|---|
| <i>object</i> | Required. A valid object.   |
| <i>String</i> | Optional. A string expression specifying the placeholder character. |

### Remarks

You can use the **PasswordChar** property to protect sensitive information, such as passwords or security codes. The value of **PasswordChar** is the character that appears in a control instead of the actual characters that the user types. If you don't specify a character, the control displays the characters that the user types.

## Picture Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proPictureC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proPictureX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proPictureA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proPictureS"}

Specifies the bitmap to display on an object.

### Syntax

*object*.**Picture** = **LoadPicture**( *pathname* )

The **Picture** property syntax has these parts:

### Remarks

While designing an HTML Layout, you can use the control's [property page](#) to assign a bitmap to the **Picture** property. While running an HTML Layout, you must use the **LoadPicture** function to assign a bitmap to **Picture**.

To remove a picture that is assigned to a control, click the value of the **Picture** property in the property page and then press DELETE. Pressing BACKSPACE will not remove the picture.

**Note** For controls with captions, use the **PicturePosition** property to specify where to display the picture on the object. Use the **PictureSizeMode** property to determine how the picture fills the object. Transparent pictures sometimes have a hazy appearance. If you do not like this appearance, display the picture on an **Image** control. **Image** controls support opaque images.



# PictureAlignment Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proPictureAlignmentC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proPictureAlignmentX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proPictureAlignmentA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proPictureAlignmentS"}

Specifies the location of a background picture.

## Syntax

*object*.**PictureAlignment** [= *fmPictureAlignment*]

The **PictureAlignment** property syntax has these parts:

| Part                      | Description   |
|---------------------------|---|
| <i>object</i>             | Required. A valid object.   |
| <i>fmPictureAlignment</i> | Optional. The position where the picture aligns with the control. |

## Settings

The settings for *fmPictureAlignment* are:

| Constant                             | Value | Description              |
|--------------------------------------|-------|--------------------------|
| <i>fmPictureAlignmentTopLeft</i>     | 0     | The top-left corner.     |
| <i>fmPictureAlignmentTopRight</i>    | 1     | The top-right corner.    |
| <i>fmPictureAlignmentCenter</i>      | 2     | The center.              |
| <i>fmPictureAlignmentBottomLeft</i>  | 3     | The bottom-left corner.  |
| <i>fmPictureAlignmentBottomRight</i> | 4     | The bottom-right corner. |

## Remarks

The **PictureAlignment** property identifies which corner of the picture is the same as the corresponding corner of the control or container where the picture is used.

For example, setting **PictureAlignment** to **fmPictureAlignmentTopLeft** means that the top-left corner of the picture coincides with the top-left corner of the control or container. Setting **PictureAlignment** to **fmPictureAlignmentCenter** positions the picture in the middle, relative to the height as well as the width of the control or container.

If you tile an image on a control or container, the setting of **PictureAlignment** affects the tiling pattern. For example, if **PictureAlignment** is set to **fmPictureAlignmentUpperLeft**, the first copy of the image is laid in the upper-left corner of the control or container and additional copies are tiled from left to right across each row. If **PictureAlignment** is **fmPictureAlignmentCenter**, the first copy of the image is laid at the center of the control or container, additional copies are laid to the left and right to complete the row, and additional rows are added to fill the control or container.

**Note** Setting the **PictureSizeMode** property to **fmSizeModeStretch** overrides **PictureAlignment**. When **PictureSizeMode** is set to **fmSizeModeStretch**, the picture fills the entire control or container.

# PicturePosition Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proPicturePositionC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proPicturePositionX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proPicturePositionA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proPicturePositionS"}

Specifies the location of the picture relative to its caption.

## Syntax

*object*.**PicturePosition** [= *fmPicturePosition*]

The **PicturePosition** property syntax has these parts:

| Part                     | Description  |
|--------------------------|--|
| <i>object</i>            | Required. A valid object.                            |
| <i>fmPicturePosition</i> | Optional. How the picture aligns with its container. |

## Settings

The settings for *fmPicturePosition* are:

| Constant                            | Value | Description   |
|-------------------------------------|-------|---|
| <i>fmPicturePositionLeftTop</i>     | 0     | The picture appears to the left of the caption. The caption is aligned with the top of the picture.     |
| <i>fmPicturePositionLeftCenter</i>  | 1     | The picture appears to the left of the caption. The caption is centered relative to the picture.        |
| <i>fmPicturePositionLeftBottom</i>  | 2     | The picture appears to the left of the caption. The caption is aligned with the bottom of the picture.  |
| <i>fmPicturePositionRightTop</i>    | 3     | The picture appears to the right of the caption. The caption is aligned with the top of the picture.    |
| <i>fmPicturePositionRightCenter</i> | 4     | The picture appears to the right of the caption. The caption is centered relative to the picture.       |
| <i>fmPicturePositionRightBottom</i> | 5     | The picture appears to the right of the caption. The caption is aligned with the bottom of the picture. |
| <i>fmPicturePositionAboveLeft</i>   | 6     | The picture appears above the caption. The caption is aligned with the left edge of the picture.        |
| <i>fmPicturePositionAboveCenter</i> | 7     | The picture appears above the caption. The caption is centered below the picture (default).             |
| <i>fmPicturePositionAboveRight</i>  | 8     | The picture appears above the caption. The caption is aligned with the right edge of the picture.       |

|                                     |    |  |
|-------------------------------------|----|--|
| <i>fmPicturePositionBelowLeft</i>   | 9  | The picture appears below the caption. The caption is aligned with the left edge of the picture.                             |
| <i>fmPicturePositionBelowCenter</i> | 10 | The picture appears below the caption. The caption is centered above the picture.  |
| <i>fmPicturePositionBelowRight</i>  | 11 | The picture appears below the caption. The caption is aligned with the right edge of the picture.                            |
| <i>fmPicturePositionCenter</i>      | 12 | The picture appears in the center of the control. The caption is centered horizontally and vertically on top of the picture. |

### Remarks

The picture and the caption, as a unit, are centered on the control. If no caption exists, the picture's location is relative to the center of the control.

This property is ignored if the **Picture** property does not specify a picture.

# PictureSizeMode Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proPictureSizeModeC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proPictureSizeModeX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proPictureSizeModeA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proPictureSizeModeS"}

Specifies how to display the background picture on a control, HTML Layout, or HTML page.

## Syntax

*object*.**PictureSizeMode** [= *fmPictureSizeMode*]

The **PictureSizeMode** property syntax has these parts:

| Part                     | Description  |
|--------------------------|--|
| <i>object</i>            | Required. A valid object.  |
| <i>fmPictureSizeMode</i> | Optional. The action to take if the picture and the HTML Layout or HTML page that contains it are not the same size. |

## Settings

The settings for *fmPictureSizeMode* are:

| Constant                        | Value | Description   |
|---------------------------------|-------|---|
| <i>fmPictureSizeModeClip</i>    | 0     | Crops any part of the picture that is larger than the HTML Layout or HTML page (default).   |
| <i>fmPictureSizeModeStretch</i> | 1     | Stretches the picture to fill the HTML Layout or HTML page. This setting distorts the picture in either the horizontal or vertical direction. |
| <i>fmPictureSizeModeZoom</i>    | 3     | Enlarges the picture, but does not distort the picture in either the horizontal or vertical direction.  |

## Remarks

The **fmPictureSizeModeClip** setting indicates you want to show the picture in its original size and scale. If the HTML Layout or HTML page is smaller than the picture, this setting only shows the part of the picture that fits within the HTML Layout or HTML page.

The **fmPictureSizeModeStretch** and **fmPictureSizeModeZoom** settings both enlarge the image, but **fmPictureSizeModeStretch** causes distortion. The **fmPictureSizeModeStretch** setting enlarges the image horizontally and vertically until the image reaches the corresponding edges of the container or control. The **fmPictureSizeModeZoom** setting enlarges the image until it reaches either the horizontal or vertical edges of the container or control. If the image reaches the horizontal edges first, any remaining distance to the vertical edges remains blank. If it reaches the vertical edges first, any remaining distance to the horizontal edges remains blank.

# PictureTiling Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proPictureTilingC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proPictureTilingX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proPictureTilingA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proPictureTilingS"}

Lets you tile a picture in an image control.

## Syntax

*object*.**PictureTiling** [= *Boolean*]

The **PictureTiling** property syntax has these parts:

| Part           | Description  |
|----------------|--|
| <i>object</i>  | Required. A valid object.  |
| <i>Boolean</i> | Optional. Specifies whether a picture is repeated across a background. |

## Settings

The settings for *Boolean* are:

| Value        | Description   |
|--------------|---|
| <b>True</b>  | The picture is tiled across the background.               |
| <b>False</b> | The picture is not tiled across the background (default). |

## Remarks

You can tile an image on an HTML Layout by drawing the **Image** the same size as the HTML Layout.

The tiling pattern depends on the current setting of the **PictureAlignment** and **PictureSizeMode** properties. For example, if **PictureAlignment** is set to **fmPictureAlignmentTopLeft**, the tiling pattern starts at the upper-left and repeats the picture across and down the height of the **Image**. If **PictureSizeMode** is set to **fmPictureSizeModeClip**, the tiling pattern crops the last tile if it doesn't completely fit within the **Image**.

# ProportionalThumb Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proProportionalThumbC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proProportionalThumbX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proProportionalThumbA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proProportionalThumbS"}

Specifies whether the size of the scroll box is proportional to the scrolling region or fixed.

## Syntax

*object*.**ProportionalThumb** [= *Boolean*]

The **ProportionalThumb** property syntax has these parts:

| Part           | Description  |
|----------------|--|
| <i>object</i>  | Required. A valid object.  |
| <i>Boolean</i> | Optional. Specifies whether the scroll box is proportional or fixed. |

## Settings

The settings for *Boolean* are:

| Value        | Description   |
|--------------|---|
| <b>True</b>  | The scroll box is proportional in size to the scrolling region (default). |
| <b>False</b> | The scroll box is a fixed size.   |

## Remarks

The size of a proportional scroll box graphically represents the percentage of the object that is visible in the window. For example, if 75 percent of an object is visible, the scroll box covers three-fourths of the scrolling region in the scroll bar.

If the scroll box is a fixed size, the system determines its size based on the height and width of the scroll bar.

# ScrollBars Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proScrollBarsC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proScrollBarsX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proScrollBarsA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proScrollBarsS"}

Specifies whether a control, form, or page has vertical scroll bars, horizontal scroll bars, or both.

## Syntax

*object*.**ScrollBars** [= *fmScrollBars*]

The **ScrollBars** property syntax has these parts:

| Part                | Description                                      |
|---------------------|--|
| <i>object</i>       | Required. A valid object.                        |
| <i>fmScrollBars</i> | Optional. Where scroll bars should be displayed. |

## Settings

The settings for *fmScrollBars* are:

| Constant                      | Value | Description   |
|-------------------------------|-------|---|
| <i>fmScrollBarsNone</i>       | 0     | Displays no scroll bars (default).                    |
| <i>fmScrollBarsHorizontal</i> | 1     | Displays a horizontal scroll bar.                     |
| <i>fmScrollBarsVertical</i>   | 2     | Displays a vertical scroll bar.                       |
| <i>fmScrollBarsBoth</i>       | 3     | Displays both a horizontal and a vertical scroll bar. |

## Remarks

If the **KeepScrollBarsVisible** property is **True**, any scroll bar on a form or page is always visible, regardless of whether the object's contents fit within the object's borders.

If visible, a scroll bar constrains its scroll box to the visible region of the scroll bar. It also modifies the scroll position as needed to keep the entire scroll bar visible. The range of a scroll bar changes when the value of the **ScrollBars** property changes, the scroll size changes, or the visible size changes.

If a scroll bar is not visible, then you can set its scroll position to any value. Negative values and values greater than the scroll size are both valid.

For a single-line control, you can display a horizontal scroll bar by using the **ScrollBars** and **AutoSize** properties. Scroll bars are hidden or displayed according to the following rules:

1. When **ScrollBars** is set to **fmScrollBarsNone**, no scroll bar is displayed.
2. When **ScrollBars** is set to **fmScrollBarsHorizontal** or **fmScrollBarsBoth**, the control displays a horizontal scroll bar if the text is longer than the edit region and if the control has enough room to include the scroll bar underneath its edit region.
3. When **AutoSize** is **True**, the control enlarges itself to accommodate the addition of a scroll bar unless the control is at or near its maximum size.

For a multiline **TextBox**, you can display scroll bars by using the **ScrollBars**, **WordWrap**, and **AutoSize** properties. Scroll bars are hidden or displayed according to the following rules:

1. When **ScrollBars** is set to **fmScrollBarsNone**, no scroll bar is displayed.
2. When **ScrollBars** is set to **fmScrollBarsVertical** or **fmScrollBarsBoth**, the control displays a vertical scroll bar if the text is longer than the edit region and if the control has enough room to include the scroll bar at the right edge of its edit region.
3. When **WordWrap** is **True**, the multiline control will not display a horizontal scroll bar. Most multiline controls do not use a horizontal scroll bar.

4. A multiline control can display a horizontal scroll bar if the following conditions occur simultaneously:
- The edit region contains a word that is longer than the edit region's width.
  - The control has enabled horizontal scroll bars.
  - The control has enough room to include the scroll bar under the edit region.
  - The **WordWrap** property is set to **False**.



## Selected Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proSelectedC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proSelectedX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proSelectedA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proSelectedS"}

Returns or sets the selection state of items in a **ListBox**.

### Syntax

*object*.**Selected**( *index* ) [= *Boolean*]

The **Selected** property syntax has these parts:

| Part           | Description  |
|----------------|--|
| <i>object</i>  | Required. A valid object.  |
| <i>index</i>   | Required. An integer with a range from 0 to one less than the number of items in the list. |
| <i>Boolean</i> | Optional. Specifies whether an item is selected.   |

### Settings

The settings for *Boolean* are:

| Value        | Description               |
|--------------|---------------------------|
| <b>True</b>  | The item is selected.     |
| <b>False</b> | The item is not selected. |

### Remarks

The **Selected** property is useful when users can make multiple selections. You can use this property to determine the selected rows in a multiselect list box. You can also use this property to select or deselect rows in a list from code.

The default value of this property is based on the current selection state of the **ListBox**.

For single-selection list boxes, the **Value** or **ListIndex** properties are recommended for getting and setting the selection. In this case, **ListIndex** returns the index of the selected item. However, in a multiple selection, **ListIndex** returns the index of the row contained within the focus rectangle, regardless of whether the row is actually selected.

When a list box control's **MultiSelect** property is set to *None*, only one row can have its **Selected** property set to **True**.

Entering a value that is out of range for the index does not generate an error message, but does not set a property for any item in the list.

## SelectedItem Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proSelectedItemC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proSelectedItemX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proSelectedItemA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proSelectedItemS"}

Returns or sets the currently selected **Tab** object.

### Syntax

*object*.SelectedItem [ =Object]

The **SelectedItem** property syntax has these parts:

| Part          | Description  |
|---------------|--|
| <i>object</i> | Required. A valid <b>TabStrip</b> .  |
| <i>Object</i> | Optional. The currently selected <b>Tab</b> of the specified <b>TabStrip</b> . |

### Remarks

Use the **SelectedItem** property to programmatically control the currently selected **Tab** object. For example, you can use **SelectedItem** to assign values to properties of a **Tab** object.

## SelectionMargin Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proSelectionMarginC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proSelectionMarginX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proSelectionMarginA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proSelectionMarginS"}

Specifies whether the user can select a line of text by clicking in the region to the left of the text.

### Syntax

*object*.**SelectionMargin** [= *Boolean*]

The **SelectionMargin** property syntax has these parts:

| Part           | Description  |
|----------------|--|
| <i>object</i>  | Required. A valid object.  |
| <i>Boolean</i> | Optional. Specifies whether clicking in the margin selects a line of text. |

### Settings

The settings for *Boolean* are:

| Value        | Description  |
|--------------|--|
| <b>True</b>  | Clicking in margin causes selection of text (default). |
| <b>False</b> | Clicking in margin does not cause selection of text.   |

### Remarks

When the **SelectionMargin** property is **True**, the selection margin occupies a thin strip along the left edge of a control's edit region. When set to **False**, the entire edit region can store text.

If the **SelectionMargin** property is set to **True** when a control is printed, the selection margin is also printed.

## SelLength Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proSelLengthC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proSelLengthX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proSelLengthA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proSelLengthS"}

Specifies the number of characters selected in a text box or the text portion of a combo box.

### Syntax

*object*.**SelLength** [= *Long*]

The **SelLength** property syntax has these parts:

| Part          | Description  |
|---------------|--|
| <i>object</i> | Required. A valid object.  |
| <i>Long</i>   | Optional. A numeric expression specifying the number of characters selected. For <b>SelLength</b> and <b>SelStart</b> , the valid range of settings is 0 to the total number of characters in the edit area of a <b>ComboBox</b> or <b>TextBox</b> . |

### Remarks

The **SelLength** property is always valid, even when the control does not have focus. Setting **SelLength** to a value less than zero creates an error. Attempting to set **SelLength** to a value greater than the number of characters available in a control results in a value equal to the number of characters in the control.

**Note** Changing the value of the **SelStart** property cancels any existing selection in the control, places an insertion point in the text, and sets **SelLength** to zero.

The default value, zero, means that no text is currently selected.

## SelStart Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proSelStartC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proSelStartX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proSelStartA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proSelStartS"}

Indicates the starting point of selected text, or the insertion point if no text is selected.

### Syntax

*object*.**SelStart** [= *Long*]

The **SelStart** property syntax has these parts:

| Part          | Description   |
|---------------|---|
| <i>object</i> | Required. A valid object.   |
| <i>Long</i>   | Optional. A numeric expression specifying the starting point of text selected. For <b>SelLength</b> and <b>SelStart</b> , the valid range of settings is 0 to the total number of characters in the edit area of a <b>ComboBox</b> or <b>TextBox</b> . The default value is zero. |

### Remarks

The **SelStart** property is always valid, even when the control does not have focus. Setting **SelStart** to a value less than zero creates an error. Attempting to set **SelStart** to a value greater than the number of characters available in a control results in a value equal to the number of characters in the control.

Changing the value of **SelStart** cancels any existing selection in the control, places an insertion point in the text, and sets the **SelLength** property to zero.

## SelText Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proSelTextC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proSelTextX": "1"} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proSelTextA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proSelTextS"}

Returns or sets the selected text of a control.

### Syntax

*object*.**SelText** [= *String*]

The **SelText** property syntax has these parts:

| Part          | Description   |
|---------------|---|
| <i>object</i> | Required. A valid object.                                   |
| <i>String</i> | Optional. A string expression containing the selected text. |

### Remarks

If no characters are selected in the edit region of the control, the **SelText** property returns a zero-length string. This property is valid regardless of whether the control has the focus.

## ShowDropButtonWhen Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proShowDropButtonWhenC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proShowDropButtonWhenX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proShowDropButtonWhenA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proShowDropButtonWhenS"}

Specifies when to show the drop-down button for a **ComboBox** or **TextBox**.

### Syntax

*object*.**ShowDropButtonWhen** [= *fmShowDropButtonWhen*]

The **ShowDropButtonWhen** property syntax has these parts:

| Part                        | Description   |
|-----------------------------|---|
| <i>object</i>               | Required. A valid object.   |
| <i>fmShowDropButtonWhen</i> | Optional. The circumstances under which the drop-down button will be visible. |

### Settings

The settings for *fmShowDropButtonWhen* are:

| Constant                          | Value | Description   |
|-----------------------------------|-------|---|
| <i>fmShowDropButtonWhenNever</i>  | 0     | Do not show the drop-down button under any circumstances. |
| <i>fmShowDropButtonWhenFocus</i>  | 1     | Show the drop-down button when the control has the focus. |
| <i>fmShowDropButtonWhenAlways</i> | 2     | Always show the drop-down button (default).               |

## SmallChange Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proSmallChangeC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proSmallChangeX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proSmallChangeA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proSmallChangeS"}

Specifies the amount of movement that occurs when the user clicks either scroll arrow in a **ScrollBar** or **SpinButton**.

### Syntax

*object*.**SmallChange** [= *Long*]

The **SmallChange** property syntax has these parts:

| Part          | Description  |
|---------------|--|
| <i>object</i> | Required. A valid object.  |
| <i>Long</i>   | Optional. An integer that specifies the amount of change to the <b>Value</b> property. |

### Remarks

The **SmallChange** property does not have units.

Any integer is an acceptable setting for this property. The recommended range of values is from –32,767 to +32,767. The default value is 1.



# SpecialEffect Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proSpecialEffectC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proSpecialEffectX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proSpecialEffectA"} HLP95EN.DLL, DYNALINK, "Specifics": "f3proSpecialEffects"} {ewc

Specifies the visual appearance of an object.

## Syntax

For **CheckBox**, **OptionButton**, **ToggleButton**

*object*.**SpecialEffect** [= *fmButtonEffect*]

For other controls

*object*.**SpecialEffect** [= *fmSpecialEffect*]

The **SpecialEffect** property syntax has these parts:

| Part                   | Description  |
|------------------------|--|
| <i>object</i>          | Required. A valid object.  |
| <i>fmButtonEffect</i>  | Optional. The desired visual appearance for a <b>CheckBox</b> , <b>OptionButton</b> , or <b>ToggleButton</b> .                     |
| <i>fmSpecialEffect</i> | Optional. The desired visual appearance of an object other than a <b>CheckBox</b> , <b>OptionButton</b> , or <b>ToggleButton</b> . |

## Settings

The settings for *fmSpecialEffect* are:

| Constant                     | Value | Description   |
|------------------------------|-------|---|
| <i>fmSpecialEffectFlat</i>   | 0     | Object appears flat, distinguished from the surrounding form by a border, a change of color, or both. Default for <b>Image</b> and <b>Label</b> , valid for all controls.   |
| <i>fmSpecialEffectRaised</i> | 1     | Object has a highlight on the top and left and a shadow on the bottom and right. Not valid for check boxes or option buttons.   |
| <i>fmSpecialEffectSunken</i> | 2     | Object has a shadow on the top and left and a highlight on the bottom and right. The control and its border appear to be carved into the form that contains them. Default for <b>CheckBox</b> and <b>OptionButton</b> , valid for all controls (default). |
| <i>fmSpecialEffectEtched</i> | 3     | Border appears to be carved around the edge of the control. Not valid for check boxes or option buttons.  |
| <i>fmSpecialEffectBump</i>   | 6     | Object has a ridge on the bottom and right and appears flat on the top and left. Not valid for check boxes or option buttons.   |

For a **Frame**, the default value is *Sunken*.

Note that only *Flat* and *Sunken* (0 and 2) are acceptable values for **CheckBox**, **OptionButton**, and **ToggleButton**. All values listed are acceptable for other controls.

## Remarks

You can use either the **SpecialEffect** or the **BorderStyle** property to specify the edging for a control, but not both. If you specify a nonzero value for one of these properties, the system sets the value of the other property to zero. For example, if you set **SpecialEffect** to **fmSpecialEffectRaised**, the system sets **BorderStyle** to zero (**fmBorderStyleNone**).

For a **Frame**, **BorderStyle** is ignored if **SpecialEffect** is **fmSpecialEffectFlat**.

**SpecialEffect** uses the system colors to define its borders.

**Note** Although the **SpecialEffect** property exists on the **ToggleButton**, the property is disabled. You cannot set or return a value for this property on the **ToggleButton**.

# Style Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proStyleC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proStyleX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proStyleA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proStyleS"}

For **ComboBox**, specifies how the user can choose or set the control's value. For **TabStrip**, identifies the style of the tabs on the control.

## Syntax

For ComboBox

*object*.**Style** [= *fmStyle*]

For TabStrip

*object*.**Style** [= *fmTabStyle*]

The **Style** property syntax has these parts:

| Part              | Description  |
|-------------------|--|
| <i>object</i>     | Required. A valid object.  |
| <i>fmStyle</i>    | Optional. Specifies how a user sets the value of a <b>ComboBox</b> . |
| <i>fmTabStyle</i> | Optional. Specifies the tab style in a <b>TabStrip</b> .             |

## Settings

The settings for *fmStyle* are:

| Constant                               | Value | Description   |
|--|-------|---|
| <i>fmStyleDropDownComb</i><br><i>o</i> | 0     | The <b>ComboBox</b> behaves as a drop-down combo box. The user can type a value in the edit region or select a value from the drop-down list (default). |
| <i>FmStyleDropDownList</i>             | 2     | The <b>ComboBox</b> behaves as a list box. The user must choose a value from the list.  |

The settings for *fmTabStyle* are:

| Constant                 | Value | Description                             |
|--------------------------|-------|---|
| <i>fmTabStyleTabs</i>    | 0     | Displays tabs on the tab bar (default). |
| <i>FmTabStyleButtons</i> | 1     | Displays buttons on the tab bar.        |
| <i>FmTabStyleNone</i>    | 2     | Does not display the tab bar.           |

## TabFixedHeight, TabFixedWidth Properties

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proTabFixedHeightC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proTabFixedHeightX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proTabFixedHeightA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proTabFixedHeightS"}

Sets or returns the fixed height or width of the tabs in points.

### Syntax

*object*.**TabFixedHeight** [= *Single*]

*object*.**TabFixedWidth** [= *Single*]

The **TabFixedHeight** and **TabFixedWidth** property syntaxes have these parts:

| Part          | Description  |
|---------------|--|
| <i>object</i> | Required. A valid object.  |
| <i>Single</i> | Optional. The number of points of the height or width of the tabs on a <b>TabStrip</b> . |

### Settings

If the value is 0, tab widths are automatically adjusted so that each tab is wide enough to accommodate its contents and each row of tabs spans the width of the control.

If the value is greater than 0, all tabs have an identical width as specified by this property.

### Remarks

The minimum size is 4 points.

## TabIndex Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proTabIndexC"} {ewc  
HLP95EN.DLL,DYNALINK,"Example":"f3proTabIndexX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies  
To":"f3proTabIndexA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proTabIndexS"}

Specifies the position of a single object in the HTML Layout's tab order.

### Syntax

*object*.**TabIndex** [= *Integer*]

The **TabIndex** property syntax has these parts:

| Part           | Description   |
|----------------|---|
| <i>object</i>  | Required. A valid object.   |
| <i>Integer</i> | Optional. An integer from 0 to one less than the number of controls on the HTML Layout that have a <b>TabIndex</b> property. Assigning a <b>TabIndex</b> value of less than 0 generates an error. If you assign a <b>TabIndex</b> value greater than the largest index value, the system resets the value to the maximum allowable value. |

### Remarks

The index value of the first object in the tab order is zero.

# TabKeyBehavior Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proTabKeyBehaviorC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proTabKeyBehaviorX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proTabKeyBehaviorA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proTabKeyBehaviorS"}

Determines whether tabs are allowed in the edit region.

## Syntax

*object*.**TabKeyBehavior** [= *Boolean*]

The **TabKeyBehavior** property syntax has these parts:

| Part           | Description                           |
|----------------|---------------------------------------|
| <i>object</i>  | Required. A valid object.             |
| <i>Boolean</i> | Optional. The effect of pressing TAB. |

## Settings

The settings for *Boolean* are:

| Value        | Description   |
|--------------|---|
| <b>True</b>  | Pressing TAB inserts a tab character in the edit region.                    |
| <b>False</b> | Pressing TAB moves the focus to the next object in the tab order (default). |

## Remarks

The **TabKeyBehavior** and **MultiLine** properties are closely related. The values described above only apply if **MultiLine** is **True**. If **MultiLine** is **False**, pressing TAB always moves the focus to the next control in the tab order regardless of the value of **TabKeyBehavior**.

The effect of pressing CTRL+TAB also depends on the value of **MultiLine**. If **MultiLine** is **True**, pressing CTRL+TAB creates a new line regardless of the value of **TabKeyBehavior**. If **MultiLine** is **False**, pressing CTRL+TAB has no effect.

# TabOrientation Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proTabOrientationC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proTabOrientationX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proTabOrientationA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proTabOrientationS"}

Specifies the location of the tabs on a **TabStrip**.

## Syntax

*object*.**TabOrientation** [= *fmTabOrientation*]

The **TabOrientation** property syntax has these parts:

| Part                    | Description                           |
|-------------------------|---------------------------------------|
| <i>object</i>           | Required. A valid object.             |
| <i>fmTabOrientation</i> | Optional. Where the tabs will appear. |

## Settings

The settings for *fmTabOrientation* are:

| Constant                      | Value | Description  |
|-------------------------------|-------|--|
| <i>fmTabOrientationTop</i>    | 0     | The tabs appear at the top of the control (default). |
| <i>fmTabOrientationBottom</i> | 1     | The tabs appear at the bottom of the control.        |
| <i>fmTabOrientationLeft</i>   | 2     | The tabs appear at the left side of the control.     |
| <i>fmTabOrientationRight</i>  | 3     | The tabs appear at the right side of the control.    |

## Remarks

If you use TrueType fonts, the text rotates when the **TabOrientation** property is set to **fmTabOrientationLeft** or **fmTabOrientationRight**. If you use bitmapped fonts, the text does not rotate.

## TabStop Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proTabStopC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proTabStopX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proTabStopA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proTabStopS"}

Indicates whether an object can receive focus when the user tabs to it.

### Syntax

*object*.**TabStop** [= *Boolean*]

The **TabStop** property syntax has these parts:

| Part           | Description   |
|----------------|---|
| <i>object</i>  | Required. A valid object.                             |
| <i>Boolean</i> | Optional. Specifies whether the object is a tab stop. |

### Settings

The settings for *Boolean* are:

| Value        | Description   |
|--------------|---|
| <b>True</b>  | Designates the object as a tab stop (default).  |
| <b>False</b> | Bypasses the object when the user is tabbing, although the object still holds its place in the actual tab order, as determined by the <b>TabIndex</b> property. |

### Remarks

The **TabStop** property can be set only at design time.



# Text Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proTextC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proTextX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proTextA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proTextS"}

Returns or sets the text in a **TextBox** or the edit area of **ComboBox**. Changes the selected row of a **ListBox**.

## Syntax

*object*.**Text** [= *String*]

The **Text** property syntax has these parts:

| Part          | Description  |
|---------------|--|
| <i>object</i> | Required. A valid object.  |
| <i>String</i> | Optional. A string expression specifying text. The default value is a zero-length string (""). |

## Remarks

For a **TextBox**, any value you assign to the **Text** property is also assigned to the **Value** property.

For a **ComboBox**, you can use **Text** to update the value of the control. If the value of **Text** matches an existing list entry, the value of the **ListIndex** property (the index of the current row) is set to the row that matches **Text**. If the value of **Text** does not match a row, **ListIndex** is set to -1.

For a **ListBox**, the value of **Text** must match an existing list entry. Specifying a value that does not match an existing list entry causes an error.

You cannot use **Text** to change the value of an entry in a **ComboBox** or **ListBox**; use the **Column** or **List** property for this purpose.

The **ForeColor** property determines the color of the text.

# TextAlign Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proTextAlignC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proTextAlignX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proTextAlignA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proTextAlignS"}

Specifies how text is aligned in a control.

## Syntax

*object*.**TextAlign** [= *fmTextAlign*]

The **TextAlign** property syntax has these parts:

| Part               | Description                                   |
|--------------------|---|
| <i>object</i>      | Required. A valid object.                     |
| <i>fmTextAlign</i> | Optional. How text is aligned in the control. |

## Settings

The settings for *fmTextAlign* are:

| Constant                 | Value | Description  |
|--------------------------|-------|--|
| <i>fmTextAlignLeft</i>   | 1     | Aligns the first character of displayed text with the left edge of the control's display or edit area (default). |
| <i>fmTextAlignCenter</i> | 2     | Centers the text in the control's display or edit area.  |
| <i>fmTextAlignRight</i>  | 3     | Aligns the last character of displayed text with the right edge of the control's display or edit area.           |

## Remarks

For a **ComboBox**, the **TextAlign** property affects only the edit region; this property has no effect on the alignment of text in the list. For stand-alone labels, **TextAlign** determines the alignment of the label's caption.

# TextColumn Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proTextColumnC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proTextColumnX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proTextColumnA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proTextColumnS"}

Identifies the column in a **ComboBox** or **ListBox** to display to the user.

## Syntax

*object*.**TextColumn** [= *Variant*]

The **TextColumn** property syntax has these parts:

| Part           | Description                           |
|----------------|---------------------------------------|
| <i>object</i>  | Required. A valid object.             |
| <i>Variant</i> | Optional. The column to be displayed. |

## Settings

Values for the **TextColumn** property range from -1 to the number of columns in the list. The **TextColumn** value for the first column is 1, the value of the second column is 2, and so on. Setting **TextColumn** to 0 displays the **ListIndex** values. Setting **TextColumn** to -1 displays the first column that has a **ColumnWidths** value greater than 0.

## Remarks

When the user selects a row from a **ComboBox** or **ListBox**, the column referenced by **TextColumn** is stored in the **Text** property. For example, you could set up a multicolumn **ListBox** that contains the names of holidays in one column and dates for the holidays in a second column. To present the holiday names to users, specify the first column as the **TextColumn**. To store the dates of the holidays, specify the second column as the **BoundColumn**.

When the **Text** property of a **ComboBox** changes (such as when a user types an entry into the control), the new text is compared to the column of data specified by **TextColumn**.

## TextLength Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proTextLengthC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proTextLengthX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proTextLengthA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proTextLengthS"}

Returns the length, in characters, of text in the edit region of a **TextBox** or **ComboBox**.

### Syntax

*object*.**TextLength** [=Long]

The **TextLength** property syntax has these parts:

| Part          | Description                                  |
|---------------|--|
| <i>object</i> | Required. A valid object.                    |
| <i>Long</i>   | The number of characters in the edit region. |

### Remarks

The **TextLength** property is read-only. For a multiline **TextBox**, **TextLength** includes LF (line feed) and CR (carriage return) characters.

# TopIndex Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proTopIndexC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proTopIndexX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proTopIndexA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proTopIndexS"}

Sets and returns the item that appears in the topmost position in the list.

## Syntax

*object*.**TopIndex** [= *Variant*]

The **TopIndex** property syntax has these parts:

| Part           | Description   |
|----------------|---|
| <i>object</i>  | Required. A valid object.   |
| <i>Variant</i> | Optional. The number of the list item that is displayed in the topmost position. The default is 0, or the first item in the list. |

## Settings

Returns the value -1 if the list is empty or not displayed.

## TripleState Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proTripleStateC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proTripleStateX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proTripleStateA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proTripleStateS"}

Determines whether a user can specify, from the user interface, the Null state for a **CheckBox**, **OptionButton**, or **ToggleButton**.

### Syntax

*object*.**TripleState** [= *Boolean*]

The **TripleState** property syntax has these parts:

| Part           | Description  |
|----------------|--|
| <i>object</i>  | Required. A valid object.  |
| <i>Boolean</i> | Optional. Specifies whether the control supports the Null state. |

### Settings

The settings for *Boolean* are:

| Value        | Description  |
|--------------|--|
| <b>True</b>  | The button clicks through three states.            |
| <b>False</b> | The button supports True and False only (default). |

### Remarks

When the **TripleState** property is **True**, a user can choose from the values of **Null**, **True**, and **False**. The null value is displayed as a shaded button.

When **TripleState** is **False**, the user can choose either **True** or **False**.

When **TripleState** is **Null**, the control does not initiate the Click event.

Regardless of the property setting, the null value can always be assigned programmatically to the button, causing the button to appear shaded.

# Value Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proValueC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proValueX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proValueA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proValueS"}

Specifies the state or content of a given control.

## Syntax

*object*.**Value** [= *Variant*]

The **Value** property syntax has these parts:

| Part           | Description                                    |
|----------------|--|
| <i>object</i>  | Required. A valid object.                      |
| <i>Variant</i> | Optional. The state or content of the control. |

## Settings

| Control                  | Description   |
|--------------------------|---|
| <b>CheckBox</b>          | An integer value indicating whether the item is selected:<br>Null Indicates the item is in a null state, neither selected nor <u>cleared</u> .<br>-1 True. Indicates the item is selected.<br>0 False. Indicates the item is cleared. |
| <b>OptionButton</b>      | Same as <b>CheckBox</b> .   |
| <b>ToggleButton</b>      | Same as <b>CheckBox</b> .   |
| <b>ScrollBar</b>         | An integer between the values specified for the <b>Max</b> and <b>Min</b> properties.   |
| <b>SpinButton</b>        | Same as <b>ScrollBar</b> .  |
| <b>ComboBox, ListBox</b> | The value in the <b>BoundColumn</b> of the currently selected rows.   |
| <b>CommandButton</b>     | Always <b>False</b> .   |
| <b>TextBox</b>           | The text in the edit region.  |

## Remarks

For a **CommandButton**, setting the **Value** property to **True** in a procedure initiates the button's Click event.

For a **ComboBox**, changing the contents of **Value** does not change the value of **BoundColumn**. To add or delete entries in a **ComboBox**, you can use the **AddItem** or **RemoveItem** method.

**Value** cannot be used with a multiselect list box.

# Visible Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "f3proVisibleC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "f3proVisibleX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "f3proVisibleA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "f3proVisibleS"}

Specifies whether a control is visible or hidden.

## Syntax

*object.Visible* [= *Boolean*]

The **Visible** property syntax has these parts:

| Part           | Description  |
|----------------|--|
| <i>object</i>  | Required. A valid object.                          |
| <i>Boolean</i> | Optional. Whether the object is visible or hidden. |

## Settings

The settings for *Boolean* are:

| Value        | Description                  |
|--------------|------------------------------|
| <b>True</b>  | Object is visible (default). |
| <b>False</b> | Object is hidden.            |

## Remarks

To hide an object at startup, set the **Visible** property to **False** at design time. Setting this property in code enables you to hide and later redisplay a control at run time in response to a particular event.

All controls are visible at design time.



## WordWrap Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"f3proWordWrapC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"f3proWordWrapX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"f3proWordWrapA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"f3proWordWrapS"}

Indicates whether the contents of a control automatically wrap at the end of a line.

### Syntax

*object*.**WordWrap** [= *Boolean*]

The **WordWrap** property syntax has these parts:

| Part           | Description  |
|----------------|--|
| <i>object</i>  | Required. A valid object.  |
| <i>Boolean</i> | Optional. Specifies whether the control expands to fit the text. |

### Settings

The settings for *Boolean* are:

| Value        | Description               |
|--------------|---------------------------|
| <b>True</b>  | The text wraps (default). |
| <b>False</b> | The text does not wrap.   |

### Remarks

For controls that support the **MultiLine** property as well as the **WordWrap** property, **WordWrap** is ignored when **MultiLine** is **False**.

**accelerator key**

A single character used as a shortcut for selecting an object. Pressing the ALT key followed by the accelerator key gives focus to the object and initiates one or more events associated with the object. The specific event or events initiated varies from one object to another. If code is associated with an event, it is processed when the event is initiated. Also called keyboard accelerator, shortcut key, keyboard shortcut.

**background color**

The color of the client region of an empty window or display screen, on which all drawing and color display takes place.

**class identifier (CLSID)**

A unique identifier (UUID) that identifies an object. An object registers its CLSID in the system registration database so the object can be loaded and programmed by other applications.

**clear**

To change a setting to "off" or remove a value.

**client region**

The portion of a window where an application displays output such as text or graphics.

**collection**

An object that contains a set of related objects. An object's position in the collection can change whenever a change occurs in the collection; therefore, the position of any specific object in the collection may vary.

**context ID**

A unique number or string that corresponds to a specific object in an application. Context IDs are used to create links between the application and corresponding Help topics.



**control group**

A set of controls that are conceptually or logically related. Controls that are conceptually related are usually viewed together but do not necessarily affect each other. Controls that are logically related affect each other. For example, setting one button in a group of option buttons sets the value of all other buttons in the group to False.

**control tip**

A brief phrase that describes a control, a **Page**, or a **Tab**. The control tip appears when the user briefly holds the mouse pointer over a control without clicking. A control tip is similar to a ToolTip. ActiveX Control Pad provides ToolTips to developers at design time, while developers provide control tips to end-users at run time.

**cursor**

A piece of software that returns rows of data to the application. A cursor on a result set indicates the current position in the result set.

**cycle**

To move through a group of objects in a defined order.

**data format**

The structure or appearance of a unit of data, such as a file, a database record, a cell in a spreadsheet, or text in a word-processing document.

**dominant control**

A reference for the **Align** command and **Make Same Size** command on the **Format** menu. When aligning controls, the selected controls align to the dominant control. When sizing controls, the selected controls are assigned the dimensions of the dominant control.

The dominant control is indicated by white sizing handles. The sizing handles of the other selected controls are black.

**drop source**

The selected text or object that is dragged in a drag-and-drop operation.

**focus**

The ability to receive mouse clicks or keyboard input at any one time. In Microsoft Windows, only one window, HTML Layout, or control can have this ability at a time. The object that "has the focus" is usually indicated by a highlighted caption or title bar. The focus can be set by the user or by the application.



**foreground color**

The color that is currently selected for drawing or displaying text on screen. In monochrome displays, the foreground color is the color of a bitmap or other graphic.

**grid block**

The space between two adjacent grid points.

**Input Method Editor (IME)**

An application that translates what you type into characters of a DBCS language, such as Japanese or Chinese. As the user types, the IME displays possible equivalents. The user selects the most appropriate entry.

**inherited property**

A property that has acquired the characteristics of another class.

**keyboard state**

A return value that identifies which keys are pressed and whether the keyboard modifiers SHIFT, CTRL, and ALT are pressed.

**OLE container control**

A Visual Basic control that is used to link and embed objects from other applications in a Visual Basic application.

**OLE object**

An object in an application that can be linked or embedded.

**OLE status code**

The error number portion of a data structure that returns information for error conditions. The data structure is defined by Object Linking and Embedding.



**placeholder**

A character that masks or hides another character for security reasons. For example, when a user types a password, an asterisk is displayed on the screen to take the place of each character typed.

**property page**

A grouping of properties presented as a tabbed page of a Properties Window.

**RGB**

A color value system used to describe colors as a mixture of red (R), green (G), and blue (B). The color is defined as a set of three integers (R,G,B) where each integer ranges from 0–255. A value of 0 indicates a total absence of a color component. A value of 255 indicates the highest intensity of a color component.

**SendKeys statement**

Sends one or more keystrokes to the active window as if typed at the keyboard.

**single-precision value**

Single (single-precision floating-point) variables are stored as IEEE 32-bit (4-byte) floating-point numbers, ranging in value from  $-3.402823E38$  to  $-1.401298E-45$  for negative values and from  $1.401298E-45$  to  $3.402823E38$  for positive values. The type-declaration character for Single is !.

**system colors**

Colors defined by the operating system for a specific type of monitor and video adapter. Each color is associated with a specific part of the user interface, such as a window title or a menu.

**target**

An object onto which the user drops the object being dragged.

**transparent**

Describes the background of the object if the background is not visible. Instead of the background, you see whatever is behind the object; for example, an image or picture used as a backdrop in your application. Use the **BackStyle** property to make the background transparent.



**z-order**

The visual layering of controls on an HTML Layout along the z-axis (depth). The z-order determines which controls are in front of other controls.



## What Is a TabStrip?

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conWhatIsTabStripC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3conWhatIsTabStripS"}

A **TabStrip** is a control that contains a collection of one or more tabs.

Each **Tab** of a **TabStrip** is a separate object that users can select. Visually, a **TabStrip** also includes a client area that all the tabs in the **TabStrip** share.

By default, a **TabStrip** includes two pages, called Tab1 and Tab2. Each of these is a **Tab** object, and together they represent the **Tabs** collection of the **TabStrip**. If you add more pages, they become part of the same **Tabs** collection.

## Tips on Using Text Boxes

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conTipsOnUsingTextBoxesC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3conTipsOnUsingTextBoxesS"}

The **TextBox** is a flexible control governed by the following properties: **Text**, **MultiLine**, **WordWrap**, and **AutoSize**.

**Text** contains the text that's displayed in the text box.

**MultiLine** controls whether the **TextBox** can display text as a single line or as multiple lines. Newline characters identify where one line ends and another begins. If **MultiLine** is **False**, then the text is truncated instead of wrapped.

**WordWrap** allows the **TextBox** to wrap lines of text that are longer than the width of the **TextBox** into shorter lines that fit.

If you do not use **WordWrap**, the **TextBox** starts a new line of text when it encounters a newline character in the text. If **WordWrap** is turned off, you can have text lines that do not fit completely in the **TextBox**. The **TextBox** displays the portions of text that fit inside its width and truncates the portions of text that do not fit. **WordWrap** is not applicable unless **MultiLine** is **True**.

**AutoSize** controls whether the **TextBox** adjusts to display all of the text. When using **AutoSize** with a **TextBox**, the width of the **TextBox** shrinks or expands according to the amount of text in the **TextBox** and the font size used to display the text.

**AutoSize** works well in the following situations:

- Displaying a caption of one or more lines.
- Displaying the contents of a single-line **TextBox**.
- Displaying the contents of a multiline **TextBox** that is read-only to the user.

**Note** Avoid using **AutoSize** with an empty **TextBox** that also uses the **MultiLine** and **WordWrap** properties. When the user enters text into a **TextBox** with these properties, the **TextBox** automatically sizes to a long narrow box one character wide and as long as the line of text.

## Create a Standard List Box

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howCreateStandardListBoxC"}  
HLP95EN.DLL,DYNALINK,"Specifics":"f3howCreateStandardListBoxS"}

{ewc

- 1 Place a **ListBox** control on an HTML Layout and select it.
- 2 In the Properties window, select the **ListStyle** property.
- 3 Click the drop-down arrow to display a list of available styles.
- 4 From the list, choose **Plain**.

## Create a List Box with Option Buttons or Check Boxes

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howCreateListBoxWithOptionButtonsOrCheckBoxesC"}  
HLP95EN.DLL,DYNALINK,"Specifics":"f3howCreateListBoxWithOptionButtonsOrCheckBoxesS"}

{ewc

- 1 Place a **ListBox** control on an HTML Layout and select it.
- 2 In the Properties window, select the **ListStyle** property.
- 3 Click the drop-down arrow to display a list of available styles.
- 4 From the list, choose **Option**.

When the **ListStyle** property is set to **Option**, the **MultiSelect** property determines whether check boxes or option buttons appear in the list. When **MultiSelect** is **Single**, option buttons appear in the list. When **MultiSelect** is **Multi** or **Extended**, check boxes appear in the list

## ListBox Styles

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conListBoxStylesC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3conListBoxStylesS"}

You can choose between two presentation styles for a **ListBox**. Each style provides different ways for users to select items in the list.

If the style is **Plain**, each item is on a separate row; the user selects an item by highlighting one or more rows.

If the style is **Option**, an option button or check box appears at the beginning of each row. With this style, the user selects an item by clicking the option button or check box. Check boxes appear only when the **MultiSelect** property is **True**.

## What Is the Difference Between the DataObject and the Clipboard?

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conDifferenceBetweenDataObjectAndClipboardC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3conDifferenceBetweenDataObjectAndClipboardS"}

The **DataObject** and the Clipboard both provide a means to move data from one place to another. As an application developer, there are several important points to remember when you use either a **DataObject** or the Clipboard:

- You can store more than one piece of data at a time on either a **DataObject** or the Clipboard as long as each piece of data has a different data format. If you store data with a format that is already in use, the new data is saved and the old data is discarded.
- The Clipboard supports picture formats and text formats. A **DataObject** currently supports only text formats.
- A **DataObject** exists only while your application is running; the Clipboard exists as long as the operating system is running. This means you can put data on the Clipboard and close an application without losing the data. The same is not true with the **DataObject**. If you close the application that put data on a **DataObject**, you lose the data.
- A **DataObject** is a standard OLE object, while the Clipboard is not. This means the Clipboard can support standard move operations (copy, cut, and paste) but not drag-and-drop operations. You must use the **DataObject** if you want your application to support drag-and-drop operations.

**Tip** You can define your own data format names when you use the **SetText** method to move data to the Clipboard or a **DataObject**. This can help distinguish between text that your application moves and text that the user moves.



## Display or Hide the Toolbox

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howDisplayOrHideToolboxC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3howDisplayOrHideToolboxS"}

On the **View** menu, determine whether a check mark appears in front of **Toolbox**. If the check mark is present, the toolbox is displayed. If not, the toolbox is hidden.

Do one of the following:

- To display the toolbox, make sure a check mark appears in front of **Toolbox**. If not, select **Toolbox**.
- To hide the toolbox, make sure there is no check mark in front of **Toolbox**. If there is, select **Toolbox** to remove it.

## What Is the Toolbox?

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conWhatIsToolboxC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3conWhatIsToolboxS"}

The toolbox identifies the different controls that you can add to an HTML Layout.

You can customize the toolbox in many ways including the following:

- Add pages to the toolbox.
- Move controls from one page to another.
- Rename pages.
- Add other controls, including ActiveX controls, to the toolbox.
- Copy customized controls from the HTML Layout into the toolbox.

For example, **OK** and **Cancel** buttons are special cases of a **CommandButton**. If you add **OK** and **Cancel** templates to the toolbox, you can quickly add them to other HTML Layouts.

## Create a Customized Control and Add It to the Toolbox

```
{ewc HLP95EN.DLL,DYNALINK,"See Also": " f3howAddCustomizedControlToToolboxC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics": " f3howAddCustomizedControlToToolboxS"}
```

**1** Place a control on your HTML Layout and customize it.

For example, to create an **OK** button, place a **CommandButton** on the HTML Layout and set its **Caption** property to **OK**.

**2** Select the customized control.

**3** Drag the control to the toolbox.

**Note** When you drag a control onto the toolbox, you transfer only property values. Any code you have written for that control is not transferred with the control. You must write new code for the icon or copy code from the control on the HTML Layout to the control on the toolbox.

## Add a Control to the Toolbox

{ewc HLP95EN.DLL,DYNALINK,"See Also": "f3howAddControlToToolboxC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics": "f3howAddControlToToolboxS"}

- 1 Right-click any control icon in the toolbox, or an empty area on any page of the toolbox.
- 2 From the shortcut menu, select **Additional Controls**.
- 3 From the **Available Controls** list, select the new controls.
- 4 Click OK.

## Add a New Item to the Toolbox

```
{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howAddNewItemToToolboxC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3howAddNewItemToToolboxS"}
```

**1** Place a control on an HTML Layout and customize it.

For example, to create an **OK** button, place a **CommandButton** on an HTML Layout, set its **Caption** property to **OK** and set its **Default** property to **True**.

**2** Select the customized control.

**3** Drag the control to the toolbox.

**Note** When you drag a control onto the toolbox, you transfer property values only . Any code you have written for that control is not transferred with the control. You must write new code for the icon or copy code from the control on the HTML Layout to the control on the toolbox.

## Add ActiveX Controls to the Toolbox

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howAddCustomControlsToToolboxC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3howAddCustomControlsToToolboxS"}

- 1 Right-click any control icon in the toolbox, or an empty area on any page of the toolbox.
- 2 From the shortcut menu, select **Additional Controls**.
- 3 From the **Additional Controls** list, select the new controls.
- 4 Click OK.

**Note** ActiveX Control Pad supports certified ActiveX controls (such as those that are certified to work with Visual Basic). The behavior of uncertified custom controls might produce unreliable results.

## Delete an Item from the Toolbox

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howDeleteItemFromToolboxC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3howDeleteItemFromToolboxS"}

- 1 In the toolbox, right-click the icon of the item you want to remove.
- 2 From the shortcut menu, select **Delete**. The command will include the name of the selected control.

**Note** If you are deleting controls, you can use **Additional Controls** from the shortcut menu, and clear the check boxes of all controls you want to delete.

## Customize a Toolbox Icon

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howCustomizeToolboxIconC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3howCustomizeToolboxIconS"}

- 1 Right-click the icon in the toolbox.
- 2 From the shortcut menu, choose **Customize**.
- 3 Do one of the following:
  - To change the ToolTip, enter the new text for the ToolTip.
  - To edit the icon, choose **Edit Picture**. Then choose the color you want to use and choose the pixel in the image where you want to apply that color.
  - To assign a new image, choose **Load Picture**. Then identify the file that contains the image you want to use as the icon. If you attempt to load a picture that is larger than the icon, an error occurs.



## What Is a ToolTip?

```
{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conWhatsToolTipC"}      {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3conWhatsToolTipS"}
```

A ToolTip is a short description, usually just a few words, that appears when the user holds the mouse pointer briefly over a control or another part of the user interface without clicking. You can customize ToolTips for controls and for the toolbox.

The default value for a new control that is copied from an HTML Layout to the toolbox is “New” followed by the control type. For example, the default ToolTip for a customized CommandButton (such as OK) is “New CommandButton”. If a control has no associated ToolTip, “Unknown” is the default value.

## Customize a ToolTip in the Toolbox

{ewc HLP95EN.DLL,DYNALINK,"See Also": "f3howCustomizeToolTipInToolboxC"}  
HLP95EN.DLL,DYNALINK,"Specifics": "f3howCustomizeToolTipInToolboxS"}

{ewc

- 1 Select the control in the toolbox.
- 2 Right-click.
- 3 From the shortcut menu, choose **Customize**. The Customize command will include the name of the control, such as "Customize Label."
- 4 Enter the new text for the ToolTip.
- 5 Click OK.

## Set the ToolTip for a Toolbox Page

{ewc HLP95EN.DLL,DYNALINK,"See Also": "f3howSetToolTipForPageOfToolboxC"}  
HLP95EN.DLL,DYNALINK,"Specifics": "f3howSetToolTipForPageOfToolboxS"}

{ewc

- 1 Select the page of the toolbox.
- 2 Right-click.
- 3 From the shortcut menu, choose **Rename**.
- 4 Enter the new text for the ToolTip.
- 5 Click OK.

## Change the Name of a Toolbox Page

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howChangeNameOfToolboxPageC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3howChangeNameOfToolboxPageS"}

- 1 Right-click the tab of the toolbox page whose name you want to change.
- 2 From the shortcut menu, choose **Rename**.
- 3 In the **Caption** field, enter the name you want to use.
- 4 Click OK.

## Change the Order of Toolbox Pages

{ewc HLP95EN.DLL,DYNALINK,"See Also": "f3howChangeOrderOfToolboxPagesC"}  
HLP95EN.DLL,DYNALINK,"Specifics": "f3howChangeOrderOfToolboxPagesS"}

{ewc

- 1 Right-click the tab of any toolbox page.
- 2 From the shortcut menu, choose **Move**.
- 3 Select the name of a page you want to move.
- 4 Choose **Move Up** or **Move Down** until the page is at the appropriate position in the page list.
- 5 Click OK.

## Create a New Toolbox Page

{ewc HLP95EN.DLL,DYNALINK,"See Also": "f3howCreateNewToolboxPageC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics": "f3howCreateNewToolboxPageS"}

- 1 Right-click the tab of any toolbox page. The new page will be inserted after this page.
- 2 Choose **New Page**.

## Delete a Toolbox Page

{ewc HLP95EN.DLL,DYNALINK,"See Also": "f3howDeleteToolboxPageC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics": "f3howDeleteToolboxPageS"}

- 1 Right-click the tab of the toolbox page you want to delete.
- 2 Choose **Delete Page**. All controls on the page are deleted at the same time.

## Import or Export a Toolbox Page

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howImportOrExportToolboxPageC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3howImportOrExportToolboxPageS"}

- 1 Right-click the tab of any page in the toolbox. If you import a page, it will be inserted after this page.
- 2 Do one of the following:
  - To import a page, choose **Import Page**. Then select the name of the page file you want to import.
  - To export a page, choose **Export Page**. Then enter a name for the file that will store a copy of the toolbox page. Exporting a page does not remove it from the toolbox.



## Move an Item to Another Toolbox Page

{ewc HLP95EN.DLL,DYNALINK,"See Also": "f3howMoveItemToAnotherToolboxPageC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics": "f3howMoveItemToAnotherToolboxPageS"}

- 1 Select a control on any page of the toolbox.
- 2 Drag the control to the tab of the new page. Hold the mouse pointer over the tab until the page appears at the front of the toolbox.
- 3 Drag the control onto the main region of the page.

**Note** If the page you want to place the control on is not visible, you can increase the width of the toolbox to display tabs for all the pages, and then drag the control to the appropriate page.

## Change the Size of the Toolbox

{ewc HLP95EN.DLL,DYNALINK,"See Also": "f3howChangeSizeOfToolboxC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics": "f3howChangeSizeOfToolboxS"}

- 1** Move the mouse pointer over an edge or a corner of the toolbox.
- 2** When the double-ended arrow appears, drag the toolbox to change its size.

## Custom Help Files

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conCustomHelpFilesC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3conCustomHelpFilesS"}

As an application developer, you can use a custom Help file to provide information about how to use your form application.

To create a custom Help file, use a product or tool that creates Windows Help files.

You can associate a specific topic in your custom Help file with each control in your application. When your application is running, the user can view your Help topic by selecting the control and pressing F1.

## Assign an Accelerator Key

{ewc HLP95EN.DLL,DYNALINK,"See Also": "f3howAssignAcceleratorKeyC"}  
HLP95EN.DLL,DYNALINK,"Specifics": "f3howAssignAcceleratorKeyS"}

{ewc

- 1 In design mode, select the control on the HTML Layout.
- 2 In the Properties window, select the **Accelerator** property.
- 3 Enter a single character as the value for **Accelerator**.

**Tip** Use a character from the caption of the control. Note that the selected character is underlined in the control's caption.

## Assign an Accelerator Key for a Tab

{ewc HLP95EN.DLL,DYNALINK,"See Also": "f3howAssignAcceleratorKeyForPageOrTabC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics": "f3howAssignAcceleratorKeyForPageOrTabS"}

- 1** In design mode, select an individual **Tab**. Be sure to select the **Tab**, not the associated **TabStrip**. When a **Tab** is selected, a rectangle appears around the caption of the **Tab**.
- 2** Right-click the selected **Tab**.
- 3** From the shortcut menu, choose **Rename**.
- 4** In the **Rename** dialog box, enter a single character in the **Accelerator Key** field.

**Tip** Use a character from the caption of the control. Note that the selected character is underlined in the control's caption.

## Assign a Caption

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howAssignCaptionC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3howAssignCaptionS"}



### To assign a caption to a **CheckBox**, **CommandButton**, **Label**, **OptionButton**, or **ToggleButton**

- 1 Display the control's Properties window.
- 2 Select the **Caption** property.
- 3 Enter the text you want to use as the caption.



### To assign a caption to a **Tab**

- 1 Select the **TabStrip** that contains the **Tab**.
- 2 Select the individual **Tab**. When the **Tab** is selected, a rectangle appears around its caption.
- 3 Right-click the selected **TabStrip**.
- 4 From the shortcut menu, choose **Rename**.
- 5 In the **Caption** field, enter the text you want to use as the caption.
- 6 Click OK.

## What Is a Caption?

```
{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conWhatIsCaptionC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3conWhatIsCaptionS"}
```

A caption is descriptive text that appears directly on or around a control. The following controls can have captions: **CheckBox**, **CommandButton**, **Label**, **OptionButton**, and **ToggleButton**. The **Tab** objects that are part of the **TabStrip** can also have captions.

## Set the Tab Order Using the TabIndex Property

{ewc HLP95EN.DLL,DYNALINK,"See Also": "f3howSetTabOrderUsingTabIndexPropertyC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics": "f3howSetTabOrderUsingTabIndexPropertyS"}

- 1 Identify the tab order you want to use for the HTML Layout.  
The tab index of the first control in the tab order is 0; the tab index of the second is 1, and so on.
- 2 Select a control in the tab order.
- 3 In the Properties window, select the **TabIndex** property.
- 4 Enter the appropriate number to identify the control's position in the tab order.



## Change the Order of Pages in a TabStrip

{ewc HLP95EN.DLL,DYNALINK,"See Also":":f3howChangeOrderOfPagesInMultiPageOrTabStripC"}  
HLP95EN.DLL,DYNALINK,"Specifics":":f3howChangeOrderOfPagesInMultiPageOrTabStripS"}

{ewc

- 1 Select any tab in the **TabStrip**.
- 2 Right-click the caption of the tab.
- 3 From the shortcut menu, choose **Move**.
- 4 In the **Move** dialog box, select the tab you want to move.
- 5 Choose **Move Up** or **Move Down** to change the position of the page.
- 6 When you've made all changes you want to, click OK.

**Note** You can also use the **Index** property to change the page order through the Properties window. The index of the first page is 0; the index of the second page is 1, and so on.

## Change the Size of the HTML Layout

{ewc HLP95EN.DLL,DYNALINK,"See Also": "f3howChangeSizeOfFormC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics": "f3howChangeSizeOfFormS"}



### To change the size of the HTML Layout at design time

- Drag the sizing handle of the HTML Layout until the HTML Layout is the size you want.



### To change the size of the HTML layout at run time

- Set the HTML Layout's **Height** and **Width** properties to the appropriate values.

## Change the Location of the HTML Layout

```
{ewc HLP95EN.DLL,DYNALINK,"See Also": "f3howChangeLocationOfFormC"}  
HLP95EN.DLL,DYNALINK,"Specifics": "f3howChangeLocationOfFormS"}
```

```
{ewc
```








### **To change the location of the HTML Layout through the user interface**

- Drag the title bar until the HTML Layout is where you want it.

## Ways to Protect Sensitive Information

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conWaysToProtectSensitiveInformationC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3conWaysToProtectSensitiveInformationS"}

Many applications use data that should be available only to certain users. Here are some suggestions you can use to protect sensitive information in ActiveX Control Pad applications:

- Write code that makes a control (and its data) invisible to unauthorized users. The **Visible** property makes a control visible or invisible. For more information about **Visible**, click .
- Write code that sets the control's foreground and background to the same color when unauthorized users run the application. This hides the information from unauthorized users. The **ForeColor** and **BackColor** properties determine the foreground color and the background color. For information about **ForeColor**, click . For information about **BackColor**, click .
- Disable the control when unauthorized users run the application. The **Enabled** property determines when a control is disabled. For information about **Enabled**, click .
- Require a password for access to the application or a specific control. You can use placeholders as the user types each character. The **PasswordChar** property defines placeholder characters. For information about **PasswordChar**, click .

**Note** Using passwords or any other techniques listed can improve the security of your application, but they do not guarantee the prevention of unauthorized access to your data.

## Make a Control That Automatically Adjusts to the Size of Its Data

```
{ewc HLP95EN.DLL,DYNALINK,"See Also": "f3howMakeControlThatAutomaticallyAdjustsC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics": "f3howMakeControlThatAutomaticallyAdjustsS"}
```

- In the Properties window, set the **AutoSize** property to **True**.

## Ways to Change the Appearance of a Control

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conWaysToChangeAppearanceOfControlC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3conWaysToChangeAppearanceOfControlS"}

ActiveX Control Pad includes several properties that let you define the appearance of controls in your application:

- **ForeColor**
- **BackColor, BackStyle**
- **BorderColor, BorderStyle**
- **SpecialEffect**

**ForeColor** determines the foreground color. The foreground color applies to any text associated with the control, such as the caption or the control's contents.

**BackColor** and **BackStyle** apply to the control's background. The background is the area within the control's boundaries, such as the area surrounding the text in a control, but not the control's border. **BackColor** determines the background color. **BackStyle** determines whether the background is transparent. A transparent control background is useful if your application design includes a picture as the main background and you want to see that picture through the control.

**BorderColor, BorderStyle, and SpecialEffect** apply to the control's border. You can use **BorderStyle** or **SpecialEffect** to choose a type of border. Only one of these two properties can be used at a time. When you assign a value to one of these properties, the system sets the other property to **None**. **SpecialEffect** lets you choose one of several border styles, but only lets you use system colors for the border. **BorderStyle** supports only one border style, but lets you choose any color that is a valid setting for **BorderColor**. **BorderColor** specifies the color of the control's border, and is only valid when you use **BorderStyle** to create the border.

## Things You Can Do with a Picture on an Image Control

```
{ewc HLP95EN.DLL,DYNALINK,"See Also": "f3conThingsPictureOnImageC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics": "f3conThingsPictureOnImageS"}
```

An **Image** control is not a picture itself; rather, it contains a picture that is stored in a separate file. You cannot edit the picture with the properties of the **Image**, but you can use them to specify the way the picture appears on the **Image**.

An interesting application of **Image** is that you can use it as a background picture for your application. To do this, make the **Image** the same size as the HTML Layout. Then, you can place other controls on top of the background.

## Align Text in a Control

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howAlignTextInControlC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3howAlignTextInControlS"}

- 1 In the Properties window, choose the **TextAlign** property.
- 2 Click the drop-down arrow next to the property's value to display a list of available choices.
- 3 Choose one of the following:
  - **Left**—to align the text with the left edge of the control.
  - **Right**—to align the text with the right edge of the control.
  - **Center**—to center the text relative to the length of the control.

**TextAlign** is available for a **ComboBox**, **Label**, and **TextBox**.



## Show or Hide the Grid

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howShowHideGridC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3howShowHideGridS"}

- Do one of the following:
  - From the **View** menu, check the **Show Grid** box to show the grid.
  - From the **View** menu, clear the **Show Grid** box to hide the grid.

**Note** To show or hide the grid by default, use the 2D Layout Options dialog box.

## Size to Grid

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howSizeToGridC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3howSizeToGridS"}

- 1 Select the control.
- 2 From the **Format** menu, choose **Size to Grid**.

The size of the selected control is adjusted so that each corner is aligned with a grid point.

## Size to Fit

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howSizeToFitC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3howSizeToFitS"}

- 1 Select the control.
- 2 From the **Format** menu, choose **Size to Fit**.

The size of the control is set so it is just large enough to display its picture and any text assigned to the **Caption** or **Text** property.

## Make Controls the Same Size

{ewc HLP95EN.DLL,DYNALINK,"See Also": "f3howMakeControlsSameSizeC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics": "f3howMakeControlsSameSizeS"}

- 1 Select all the controls you want to be the same size.
- 2 Select the dominant control.
- 3 From the **Format** menu, point to **Make Same Size**, and then click one of the following:
  - **Width**—to make all selected controls the same width as the dominant control.
  - **Height**—to make all selected controls the same height as the dominant control.
  - **Both**—to make all selected controls the same height and width as the dominant control.

## Align Controls

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howAlignControlsC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3howAlignControlsS"}

- 1 Select the controls to align.
- 2 Select the dominant control.
- 3 From the **Format** menu, point to **Align**, and then click one of the following to align the specified part of each selected control with the same part of the dominant control:
  - **Lefts**—to align the left edge.
  - **Centers**—to align the center of each control. This means a vertical line drawn at the center of the dominant control would contain the center of every selected control.
  - **Rights**—to align the right edge.
  - **Tops**—to align the top.
  - **Middles**—to align the center of each control. This means a horizontal line drawn at the center of the dominant control would also contain the center of every selected control.
  - **Bottoms**—to align the bottom.
  - **To Grid**—to align the upper-left corner of each selected control with its nearest grid point. Note that this option is not based on the position of the dominant control.

**Note** Each command on the menu has a small picture that shows how the controls will be aligned.

## Adjust Horizontal and Vertical Spacing Between Controls

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howAdjustHorizontalSpacingC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3howAdjustHorizontalSpacingS"}






- 1 Select the controls where you want to adjust spacing.
- 2 From the **Format** menu, point to **Horizontal Spacing** or **Vertical Spacing**, and then click one of the following:
  - **Make Equal**—to make all horizontal and vertical spaces between controls the same size. The amount of horizontal and vertical space will vary depending on the area available for displaying controls and the combined width of all controls.
  - **Increase**—to increase the space between controls by one grid block.
  - **Decrease**—to decrease the space between controls by one grid block.
  - **Remove**—to remove the space between controls. The controls do not overlap, but are immediately adjacent to each other.

## Things You Can Do with Control Groups

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conThingsControlGroupsC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3conThingsControlGroupsS"}

A group is two or more controls on an HTML Layout that you treat as a single unit. You can include any control on the HTML Layout in a group. Once controls belong to a group, you can work with the entire group, or you can select a single control.


ActiveX Control Pad provides many ways to work with groups and the controls in a group. After you select a group, you can do any of the following:

- Size all controls in the group at the same time. For more information, click .
- Select a single control inside a group. For more information, click .
- Break up the group so each control is independent of the others. For more information, click .
- Display the group's shortcut menu, which provides quick access to commands that affect the group. For more information, click .
- Select a single control within the group without breaking up the group, which lets you change property settings of that control without affecting any other control in the group. For more information, click .

## Transparency in ActiveX Control Pad

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conTransparencyInMSFormsC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3conTransparencyInMSFormsS"}

ActiveX Control Pad supports transparency in two areas: the background of certain controls, and in images used on certain controls.

The **BackStyle** property determines whether a control is transparent. A transparent control lets you see what is behind it on the HTML Layout. This is useful if you have a decorative background on the HTML Layout and you want to minimize the amount of that background that is hidden behind the controls. For more information on making a control transparent, click .

You can display an image on many controls in ActiveX Control Pad. Certain controls support transparent images, that is, images in which one or more background color is transparent. Image transparency is not controlled by any control property; it is controlled by the color of the lower-left pixel in the image. ActiveX Control Pad does not provide a way to edit an image and make it transparent; you must use a picture editor for this purpose.

In ActiveX Control Pad, images are always transparent on the following controls:

- **CheckBox**
- **CommandButton**
- **Label**
- **OptionButton**
- **ToggleButton**

Transparent pictures sometimes have a hazy appearance. If you do not like this appearance, display the picture on an **Image** control. **Image** controls support opaque images.

If you use a transparent image on a control that does not support transparent images, the image will be displayed correctly but you won't be able to see what's behind it. In ActiveX Control Pad, the **Image** control does not support transparent images.



## What is a Shortcut Menu?

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conWhatIsContextMenuC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3conWhatIsContextMenuS"}

A shortcut menu is a menu that appears when you right-click an object. In ActiveX Control Pad, the following objects have shortcut menus:

- The toolbox, each page in the toolbox, and each item on a page of the toolbox.
- Individual controls on an HTML Layout.
- Groups of controls (groups created with the **Group** command).
- Containers (such as UserForm).
- Individual **Tab** objects in a **TabStrip**.
- Multiple controls that aren't in a group.

The commands on a shortcut menu vary depending on the object you select. For example, if you select multiple controls that aren't in a group, the shortcut menu will include the **Group** command; the shortcut menu for the toolbox will not.

To display the shortcut menu for a control or container, right-click the object.

For more information on displaying the shortcut menu for a **TabStrip** or a **Tab**, click .

## Ways to Put Data in a ListBox or ComboBox

```
{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conWaysToPutDataInListC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3conWaysToPutDataInListS"}
```

In a **ListBox** or **ComboBox** with a single column, the **AddItem** method provides an effective technique for adding an individual entry to the list. In a multicolumn **ListBox** or **ComboBox**, however, the **List** and **Column** properties offer another technique; you can load the list from a two-dimensional array.

## Things You Can Do with a Multicolumn ListBox or ComboBox

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conThingsYouCanDoWithListC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3conThingsYouCanDoWithListS"}

To control the column widths of a multicolumn **ListBox** or **ComboBox**, you can specify the width, in points, for all the columns in the **ColumnWidths** property. Specifying zero for a specific column hides that column of information from the display.

If you want to hide all but one column of a **ListBox** or **ComboBox** from the user, you can identify the column of information to display by using the **TextColumn** property.

## Add Items to a List Using the List or Column Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howAddItemsToListUsingListOrColumnPropertyC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3howAddItemsToListUsingListOrColumnPropertyS"}

- 1 Create a multicolumn **ListBox** or **ComboBox**.
- 2 Create a two-dimensional array that contains the items you want to put in the list.
- 3 Set the **ColumnCount** property of the **ListBox** or **ComboBox** to match the number of entries in the list.
- 4 Do one of the following:
  - Assign the array as the value of the **List** property. The contents of the **ListBox** will match the contents of the array exactly.
  - Assign the array as the value of the **Column** property. **Column** transposes rows and columns, so each row of the **ListBox** matches the corresponding column of the array.

## Object Model for ActiveX Control Pad

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conObjectModelMSFormsC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3conObjectModelMSFormsS"}

The ActiveX Control Pad object model includes the following types of objects:

- Controls
- Objects (within collections)

Each element of the ActiveX Control Pad object model has some combination of properties, events, and methods that you can use to make your application work the way you want it to.

ActiveX Control Pad has three collections:

**Controls** collection—contains all the controls on a form.

**Tabs** collection—contains all the **Tab** objects in a **TabStrip**. Each **TabStrip** has its own distinct **Tabs** collection.

## Creating an Option Group

```
{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conWaysToCreateOptionGroupC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3conWaysToCreateOptionGroupS"}
```

By default, all **OptionButton** controls on an HTML Layout are part of a single option group. This means that selecting one of the buttons automatically sets all other option buttons on the HTML Layout to **False**.

If you want more than one option group on the HTML Layout, there are two ways to create additional groups use the **GroupName** property to identify related buttons.

**Note** A **TabStrip** is not a container. Option buttons in the **TabStrip** are included in the HTML Layout's option group. You can use **GroupName** to create an option group from buttons in a **TabStrip**.

## Create an Option Group Using the `GroupName` property

```
{ewc HLP95EN.DLL,DYNALINK,"See Also": "f3howCreateOptionGroupUsingGroupNameC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics": "f3howCreateOptionGroupUsingGroupNameS"}
```

- 1 Place all required **OptionButton** controls on the HTML Layout.
- 2 Identify the buttons for each group you want to create.
- 3 Enter a value for the **ID** property of each control.
- 4 For each button in a group, set the **GroupName** property to the same value.

## Ways to Match Entries in a List

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conWaysToMatchEntriesInListC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3conWaysToMatchEntriesInListS"}

There are three ways to match a value entered by the user with an entry that exists in the list of a **ListBox** or **ComboBox**:

- **No matching**—provides no assistance in matching a user's typed entry to an entry in the list.
- **First letter**—compares the most recently-typed letter to the first letter of each entry in the list. The first match in the list is selected.
- **Complete**—compares the user's entry and tries to find an exact match in an entry from the list.

The matching feature resets after two seconds (six seconds in the Far East version). For example, if you have a list of the 50 states and you type "CO" quickly, you will find "Colorado." But if you type "CO" slowly, you will find "Ohio" because the auto-complete search resets between letters.

If you choose **Complete** matching, it is a good idea to sort the list entries alphabetically (you can use the **TextColumn** property to do this). If the list is not sorted alphabetically, matching may not work correctly. For example, if the list includes Alabama, Louisiana, and Alaska in that order, then "Alabama" will be considered a complete match if the user types "ala." In fact, this result is ambiguous because there are two entries in the list that could match what the user entered. Sorting alphabetically eliminates this ambiguity.



## Use Z-order to Layer Controls

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howLayerControlsC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3howLayerControlsS"}



### To place a control at the front or back of the z-order

- 1 Select a control whose z-order you want to reposition.
- 2 From the **Format** menu, click **Bring to Front** or **Send to Back**.



### To adjust a control one position in the z-order

- 1 Select the control you want to reposition.
- 2 From the **Format** menu, click **Move Forward** or **Move Backward**.

**Note** You can't Undo or Redo layering commands, such as **Send to Back** or **Bring to Front**. For example, if you select an object and click **Move Backward** on the shortcut menu, you won't be able to Undo or Redo that action.

If the form includes any **ListBox** controls, those controls automatically move as close as possible to the top of the stack. The **Bring to Front**, **Move Forward**, **Send to Back**, and **Move Backward** menu choices let you change the z-order of a control relative to other similar controls. For example, applying **Move Backward** to a **ListBox**, moves the control below other **ListBox** controls, but will not move it below any other type of control in the stack. Similarly, applying **Move Forward** to a control other than a **ListBox**, will move the control closer to top of the stack, but will not move it above any **ListBox** in the stack.

## Create a Transparent Control

```
{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howCreateTransparentControlC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3howCreateTransparentControlS"}
```

- 1 Put the basic control onto the HTML Layout.
- 2 View the control's properties.
- 3 Set the **BackStyle** property to **Transparent**.
- 4 Set the **BorderStyle** property to **None**.
- 5 Set the **BackColor** property to **None** if you want to display only the text associated with the control, or to a valid color if you want to display a transparent block of color.

## Delete an Image from a Control

```
{ewc HLP95EN.DLL,DYNALINK,"See Also": "f3howDeleteBitmapFromControlC"}  
HLP95EN.DLL,DYNALINK,"Specifics": "f3howDeleteBitmapFromControlS"}
```

{ewc



### To delete an image using the Properties window

- 1 Highlight the value of the **Picture** property (the word "Picture").
- 2 Press DELETE.



### To delete an image using code

- Enter the following statement: **Object.Picture = LoadPicture("")**

## Assign an Image to a Control

```
{ewc HLP95EN.DLL,DYNALINK,"See Also": "f3howAssignBitmapToControlC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics": "f3howAssignBitmapToControlS"}
```

In the Properties window:

- 1 Choose the **Picture** or **PicturePath** property.
- 2 In the **Picture** dialog box, enter the name of the picture and its location.

If the picture is larger than the control, the picture is scaled to fit the control, regardless of whether you assign the picture through the Properties window or through code. The **PictureAlignment** property determines how it is aligned within the control.

## Ways to Align a Picture on a Control

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conWaysToAlignPictureOnControlC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3conWaysToAlignPictureOnControlS"}

The **Picture** (or **PicturePath**) property assigns an image or other picture to a control. After you assign the picture to the control, you can do any of the following to align the picture on the control:

- Use the **PictureAlignment** property to center the picture within the **Image** or align any corner of the picture with the corresponding corner of the **Image**.
- Use the **PictureSizeMode** property to clip, stretch, or zoom the picture within the **Image**. Stretching can distort the picture, but zooming will not.
- Use the **PictureTiling** property to display multiple copies of the picture within the **Image**.

## Select a Grid Size

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howSelectGridSizeC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3howSelectGridSizeS"}

- 1 From the **Tools** menu, choose **Options** and then click **HTML Layout**.
- 2 In the **HTML Layout Options** dialog box, specify the size you want for each grid block. Specifying smaller numbers results in smaller grid blocks.

## Create a Control Group

```
{ewc HLP95EN.DLL,DYNALINK,"See Also": "f3howCreateControlGroupC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics": "f3howCreateControlGroupS"}
```

- 1 In the HTML Layout, select each control you want to include in the group.
- 2 From the shortcut menu, choose **Group**.

## Size All the Controls in a Group

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howSizeAllControlsInGroupC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3howSizeAllControlsInGroupS"}

**1** Select the group.

A rectangle with sizing handles surrounds the group to indicate it is selected.

**2** Click one of the sizing handles and drag it to change the size of the rectangle.

**3** Release the mouse button.

The size of each control will be changed proportionately to the way you changed the rectangle around the group.



## Break Up a Control Group

```
{ewc HLP95EN.DLL,DYNALINK,"See Also":f3howBreakUpControlGroupC"}  
HLP95EN.DLL,DYNALINK,"Specifics":f3howBreakUpControlGroupS}
```

```
{ewc
```

- 1 Select the group.
- 2 From the shortcut menu, choose **Ungroup**.

## Display a Group's Shortcut Menu

```
{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howDisplayGroupsContextMenuC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3howDisplayGroupsContextMenuS"}
```

- 1 Select the group.
- 2 Right-click on a control in the group.

**Tip** To make the shortcut menu go away without selecting any of its commands, press ESC.

## Select a Control Within a Group

{ewc HLP95EN.DLL,DYNALINK,"See Also":":f3howSelectControlWithinGroupC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":":f3howSelectControlWithinGroupS"}

- 1** Select the group.
- 2** Select a single control within the group. The sizing handles around the group become lighter, and dark handles appear on the selected control.  
You can change the value of the selected control's properties. Any changes you make will affect only the selected control.
- 3** When you're finished working with the selected control, click anywhere inside the group, but not on the selected control. The group is still selected.  
You can select another control in the group or go on to any other task you need to perform.

## Display the Shortcut Menu for an HTML Layout

```
{ewc HLP95EN.DLL,DYNALINK,"See Also": "f3howDisplayContextMenuForMultiPageOrPageC"}  
HLP95EN.DLL,DYNALINK,"Specifics": "f3howDisplayContextMenuForMultiPageOrPageS"}
```

{ewc

- Right-click the Layout, not the controls.

## Display the Shortcut Menu for a TabStrip or Tab

{ewc HLP95EN.DLL,DYNALINK,"See Also": "f3howDisplayContextMenuForTabStripOrTabC"}  
HLP95EN.DLL,DYNALINK,"Specifics": "f3howDisplayContextMenuForTabStripOrTabS"}

{ewc



### To display the shortcut menu of an individual Tab

- 1 In design time, select the appropriate tab.
- 2 Right-click the selected caption.



### To display the shortcut menu of the TabStrip

- In design time, right-click anywhere in the **TabStrip** control.

If an individual **Tab** is selected, the **Tab's** shortcut menu is displayed instead. To select the **TabStrip**, click the HTML Layout anywhere outside of the **TabStrip**, and then right-click the **TabStrip** again.

## Rename Dialog Box

Contains the **Accelerator**, and **Caption** property settings for the individual tab that has the focus. Contains the caption and ToolTip text for the current toolbox page. You can update the values for these properties.

The accelerator key is a keyboard key that the user presses simultaneously with ALT to set the focus to a **Tab**. The caption is the text in the tab area of a **Tab** or the current toolbox page.



### To set an accelerator for the Tab:

- Enter a single character for **Accelerator**.



### To rename the Tab:

- Enter a new value for **Caption**.

## Tab Order Dialog Box



### To change the position of a page or tab

- 1 Select the name of the **Tab** you want to move.
- 2 Choose **Move Up** or **Move Down** until the selected item is in the desired location.
- 3 When all items are in the order you want, click OK.

## Additional Controls Dialog Box

- 1 In the **Available Controls** list, select the control or controls you want to add to the toolbox.
  - 2 Click OK.
- Tip** You can filter the **Available Controls** list by selecting options in the **Show** group.



## Customize Control Dialog Box

Contains the **ToolTipText** property and the icon that represents this control in the toolbox. With this dialog box, you can define or change the **ToolTipText** associated with this control, as well as change the icon that is displayed in the toolbox.



### To define or edit ToolTipText

- Enter a new value for **ToolTipText**.



### To edit the icon

- 1 Choose the **Edit Picture CommandButton**.
- 2 Use the **Image Editor** to alter the icon as needed.



### To load another icon

- 1 Choose the **Load Picture CommandButton**.
- 2 From the common dialog box, select a picture file.
- 3 Click OK to apply the new values.

## Compress Audio Files

Before adding audio .wav files to an HTML Layout, you might want to compress them with the MSN Audio codec to reduce their file size. One way to do this is by choosing this compression type in the Windows 95 Sound Recorder.

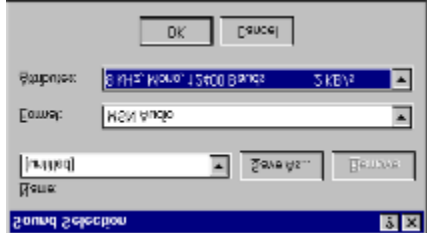


### To compress an audio .wav file

- 1 Open the file in the Windows 95 Sound Recorder.
- 2 Click the “Change...” button in the “Save As... dialog box.



- 3 In the Sound selection dialog box, choose MSN Audio.



- 4 In the **Attributes** field, select the compression setting. The best setting depends on the modem speed you are designing for.

**Note** If you choose a compression setting that is too high, you may hear breaks in the audio clip when playing it back.

Here are some suggested settings:

| <b>Modem speed, bps</b> | <b>Attributes</b>              |
|-------------------------|--------------------------------|
| 9600                    | 8kHz, Mono, 8200 Baud, 1KB/s   |
| 14,400                  | 8kHz, Mono, 12,400 Baud, 2KB/s |
| 28,800                  | 22kHz, Mono, 22602 Baud, 3KB/s |

## Active controls and selected controls

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conActiveControlsSelectedControlsC"} {ewc  
HLP95EN.DLL,DYNALINK,"Specifics":"f3conActiveControlsSelectedControlsS"}

All controls have an active state and a selected state. When a control is active, it means you are working with the contents of the control; when a control is selected, it means you are working with the control itself.

Most controls are automatically selected when you put them on the form. In design mode, sizing handles appear around a control's border when the control is selected. If you deselect the control, you can select it again by clicking once on the control.

Clicking a control that is selected puts the control in the active state. In this state, you can directly edit the control's caption.

In either the selected state or the active state, you can use DEL, CTRL+X and CTRL+C as shortcut keys for the **Delete**, **Cut**, and **Copy** commands, respectively. In the selected state, these commands are available on the shortcut menu and will affect the control itself. In the active state, these commands will affect whatever text is selected inside the control; if no text is selected, these commands have no effect. These commands are not available on the shortcut menu for active controls.

## Tips on Selecting Multiple Controls

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conTipsOnSelectingMultipleControlsC"}  
HLP95EN.DLL,DYNALINK,"Specifics":"f3conTipsOnSelectingMultipleControlsS"}

{ewc

You can select more than one control in three ways:

- **SHIFT+CLICK**: ActiveX Control Pad creates an invisible selection rectangle around the selected controls and puts sizing handles on all controls within that rectangle.
- **CTRL+CLICK**: sizing handles appear only on the selected controls, not on controls within the surrounding rectangle. If this method selects additional controls that are near to or adjacent to the selected controls, use the **Select Objects** pointer explained below.
- **Select Objects** pointer on the toolbox: draw a rectangle over the controls you want to select. All controls that fall within or just touch the rectangle will be selected.

**ANSI character set**

American National Standards Institute (ANSI) 8-bit character set used by Microsoft Windows to represent up to 256 characters (0–255) using your keyboard. The first 128 characters (0–127) correspond to the letters and symbols on a standard U.S. keyboard. The second 128 characters (128–255) represent special characters, such as letters in international alphabets, accents, currency symbols, and fractions.

**array**

A set of sequentially indexed elements having the same intrinsic data type. Each element of an array has a unique identifying index number. Changes made to one element of an array do not affect the other elements.

**class**

The formal definition of an object. The class acts as the template from which an instance of an object is created at run time. The class defines the properties of the object and the methods used to control the object's behavior.

**container**

An object that can contain other objects.



**module**

A set of declarations followed by procedures.

## **named arguments**

An argument that has a name that is predefined in the object library. Instead of providing a value for each argument in a specified order expected by the syntax, you can use named arguments to assign values in any order. For example, suppose a method accepts three arguments:

**DoSomething *namedarg1, namedarg2, namedarg3***

By assigning values to named arguments, you can use the following statement:

```
DoSomething namedarg3 := 4, namedarg2 := 5, namedarg1 := 20
```

Note that the arguments don't need to appear in their normal positional order.

## **Null**

A value indicating that a variable contains no valid data. **Null** is the result of an explicit assignment of **Null** to a variable or any operation between expressions that contain **Null**.

**point**

In typography, a point is  $\frac{1}{72}$  inch. The size of a font is usually expressed in points.

**project**

A set of modules.

**tab order**

The order in which the focus moves from one field or object to the next as you press TAB or SHIFT+TAB.



## Date and Time Example

This example uses a **CommandButton** Click event to update the **Caption** of a **Label** by choosing the value of two **CheckBoxes**.

The following controls and corresponding property values should be set:

- Add **CheckBox1**
  - Set **Caption** = "Show Date"
  - Set **Value** = 0
- Add **CheckBox2**
  - Set **Caption** = "Show Time"
  - Set **Value** = 0
- Add **Label1**
  - Set **Caption** = "Date and time displayed here"
  - Set **TextAlign** = Center
  - Set **BorderStyle** = Single
- Add **CommandButton1**
  - Set **Caption** = "Display"

For the **CommandButton1** Click event, add the following code:

```
Dim result
If CheckBox1.Value = True Then
    If CheckBox2.Value = True Then
        result = Date() & Chr(32) & Time()
    Else
        result = Date()
    End If
Else
    If CheckBox2.Value = True Then
        result = Time()
    Else
        result = "Date and time displayed here"
    End If
End If

Label1.Caption = result
```



## Form Information Example

This example uses **OptionButtons** to change the background color of an HTML Layout and a **CommandButton** to display the width and height of the HTML Layout in a dialog box.

The following controls and corresponding property values should be set:

- Add **OptionButton1**
  - Set **Caption** = "Red"
  - Set **Value** = 0
- Add **OptionButton2**
  - Set **Caption** = "Green"
  - Set **Value** = 0
- Add **OptionButton3**
  - Set **Caption** = "Blue"
  - Set **Value** = 0
- Add **CommandButton1**
  - Set **Caption** = "Size"

For the following events, add the corresponding code:

- For the **OptionButton1** Click event:  
`Form.BackColor = RGB(255,0,0)`
- For the **OptionButton2** Click event:  
`Form.BackColor = RGB(0,255,0)`
- For the **Optionbutton3** Click event:  
`Form.BackColor = RGB(0,0,255)`
- For the **CommandButton1** Click event:  
`MsgBox("HTML Layout width = " & Form.Width & chr(13) & chr(10) & _  
"HTML Layoutheight = " & Form.Height)`

## Adding Links via Colored Labels Example

This example uses **Labels** to provide links to other sites rather than underlined hypertext links. This text describes the process for adding one **Label**. This example could be extended to also use images and hot spots.

The following control and corresponding property values should be set:

- Add **Label1**
  - Select a background color and foreground color. Set Caption (for example "Microsoft").

For the **Label1** MouseDown event, add the following code:

```
Window.location.href = "http://www.microsoft.com"
```

**Mouse Down Event**

## Hello World Example

This example uses a **CommandButton** Click event to display the message "Hello World" in a dialog box.

The following control and corresponding property value should be set:

- Add **CommandButton1**
  - Set **Caption** = "Push"

For the **CommandButton1** Click event, add the following code:

```
MsgBox("Hello, World!")
```

## Hide/Show Controls Example

This example demonstrates a method of hiding and showing **CommandButtons** on an HTML Layout.

The following controls and corresponding property values should be set:

- Add **CommandButton1**
  - Set Caption = "Show the other button"
- Add **CommandButton2**
  - Set Caption = "Bring back the first button"

For the following events, add the corresponding code:

- For the **CommandButton1** Click event:

```
CommandButton2.Visible = True
CommandButton1.Visible = False
```
- For the **CommandButton2** Click event:

```
CommandButton1.Visible = True
CommandButton2.Visible = False
```

## Add/Remove Items from a ListBox Example

This example demonstrates a method for interactively updating a **ListBox**.

The following controls and corresponding property values should be set:

- Add **CommandButton1**
  - Set **Caption** = "Add Item"
- Add **CommandButton2**
  - Set **Caption** = "Remove Item"
- Add **ListBox1**

For the following events, add the corresponding code:

- For the **CommandButton1** Click event:

```
Dim NewItem
NewItem = InputBox("Enter new item to add to list box", "Add item")
NewItem = Trim(NewItem)
If Len(NewItem) > 0 Then
    ListBox1.AddItem(NewItem)
End If
```

- For the **CommandButton2** Click event

```
If ListBox1.ListIndex >= 0 Then
    ListBox1.RemoveItem(ListBox1.ListIndex)
    ListBox1.SetFocus
Else
    MsgBox("No item selected!")
End If
```

## Mouse Tracking Example

This example uses the `MouseOver` event for tracking mouse movement and updates a **Label** based on the mouse position.

The following controls and corresponding property values should be set:

- Add **Label1**
  - Set **Caption** = "Number One"
  - Set **BorderStyle** = "Single"
- Add **Label2**
  - Set **Caption** = "Number Two"
  - Set **BorderStyle** = "Single"
- Add **CommandButton1**
  - Set **Caption** = "Button 1"
- Add **Label3**
  - Set **Caption** = ""
  - Set **ID** = "lblDisplay"
  - Set **BorderStyle** = "Single"
  - Set **TextAlign** = "Center"

For the following events, add the corresponding code:

- For **Label1** `MouseDown` event:

```
lblDisplay.Caption = "Mouse down number one"
```
- For **Label1** `MouseMove` event:

```
lblDisplay.Caption = "Mouse moving over number one"
```
- For **Label2** `MouseDown` event:

```
lblDisplay.Caption = "Mouse down number two"
```
- For **Label2** `MouseMove` event:

```
lblDisplay.Caption = "Mouse moving over number two"
```
- For `lblDisplay_MouseMove` event:

```
lblDisplay.Caption = "Mouse moving over display label"
```
- For **CommandButton1** `MouseMove` event:

```
lblDisplay.Caption = "Mouse moving over command button"
```

## Resizing an Image Example

This example demonstrates dynamically resizing an image.

The following controls and corresponding property values should be set:

- Add **Image1**
  - Assign the **PicturePath** property to some file. For example: "file:///c:\windows\test.bmp".
  - Set **PictureSizeMode** = "Stretch"
  - Size the HTML Layout to approximately twice as high and twice as wide as the image.
- Add **CommandButton1**
  - Set **Caption** = "Small"
- Add **CommandButton2**
  - Set **Caption** = "Medium"
- Add **CommandButton3**
  - Set **Caption** = "Large"

For the following events, add the corresponding code:

- For **CommandButton1** Click event:

```
Image1.Width = form.Width / 4
Image1.Height = form.Height / 4
Image1.Left = (form.Width/2) - (Image1.Width/2)
Image1.Top = (form.Height/2) - (Image1.Height/2)
```
- For **CommandButton2** Click event:

```
Image1.Width = form.Width / 2
Image1.Height = form.Height / 2
Image1.Left = (form.Width/2) - (Image1.Width/2)
Image1.Top = (form.Height/2) - (Image1.Height/2)
```
- For **CommandButton3** Click event:

```
Image1.Width = form.Width
Image1.Height = form.Height
Image1.Left = 0
Image1.Top = 0
```

## SpinButton Control Updating a Label Example

This example demonstrates dynamically updating the value of a **label** with a **SpinButton**.

The following controls and corresponding property values should be set:

- Add **SpinButton1**
- Add **Label1**
  - Set **Caption** = ""
  - Set **BorderStyle** = "Single"
  - Set **TextAlign** = "Center"
- For the **SpinButton1** event, add the following code:  
`Label1.Caption = SpinButton1.Value`



## Web Browser Inside a Web Browser Example

This example uses the **Web Browser** control to open a Web page within another Web page.

The following controls and corresponding property values should be set:

- Add **Label1**
  - Set **ForeColor**, **BackColor** and **Caption** to your preference.
- Add **WebBrowser**
  - Set **ID** = "web1"
  - Size as desired
- For the **Label1** MouseDown event, add the following code:  

```
call web1.Navigate("http://www.microsoft.com")
```

## ActiveMovie Control

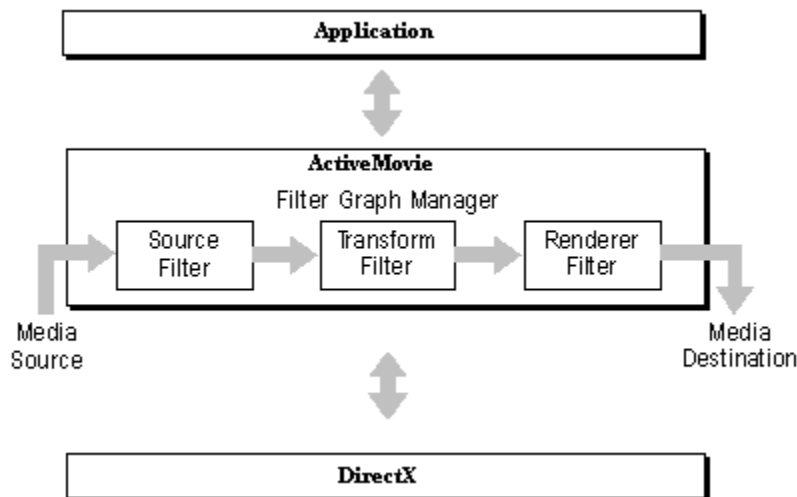
[Properties](#)   [Methods](#)   [Events](#)   [See Also](#)

The **ActiveMovie** control is a custom control that you can use with Microsoft Visual Basic and Microsoft Visual C++ to quickly add support for multimedia streams to your applications.

ActiveMovie™ is an extensible media streaming architecture for Windows that delivers high quality audio and video playback from the Internet or Intranet. ActiveMovie supports the most popular media types, including MPEG audio and video, AVI video, WAV audio, and Apple® QuickTime® video.

The ActiveMovie architecture defines how streams of time-stamped multimedia data can be controlled and processed by using modular components called *filters* connected in a configuration called a *filter graph*.

Applications assemble the filter graph and control how data moves through the filter graph by accessing the *filter graph manager* via programming interfaces, as shown in the following illustration:



For example, the Microsoft MPEG filter graph uses the following filters:

- A source filter to read the data off the disk.
- A splitter transform filter to separate the video and audio.
- A video transform filter to decompress the video data.
- A video rendering filter to display the data on the screen.
- An audio transform filter to decompress the audio data.
- An audio rendering filter to send the audio data to the sound card.

Default filter graphs are configured for you when you install the ActiveMovie software on your computer. You can also install additional filters and create your own filter graphs. For more information about configuring filter graphs, see the ActiveMovie Software Developer's Kit (SDK) documentation.

The **ActiveMovie** control represents an easy-to-use programming interface that lets you manage multimedia streams using the control's properties, methods, and events. The control handles all video and audio rendering for you, simplifying your programming tasks and making it easy to add support for multimedia streams to your application.

In addition to the **ActiveMovie** control, two other ActiveMovie programming interfaces are available:

- ActiveMovie Component Object Model (COM) interfaces
- The OM-1 MPEG MCI command set

For more information about these programming interfaces, see the ActiveMovie SDK documentation.

## See Also

[Shortcut Keys](#)

[Properties Pages](#)

## ActiveMovie Properties

See Also

The **ActiveMovie** control supports the following properties:

**Note** Some **ActiveMovie** properties are only functional with some types of multimedia streams. For example, the properties **Author**, **Copyright**, **Description**, **ImageSourceHeight**, **ImageSourceWidth**, and **Rating** are only available for ActiveMovie streaming format (ASF) files.

| <u>Property</u>                | <u>Description</u>  |
|--------------------------------|---|
| <u>AllowChangeDisplayMode</u>  | Indicates whether the end-user can change the display mode at run time between time and frames. |
| <u>AllowHideControls</u>       | Indicates whether the end-user can hide the control panel at run time.                          |
| <u>AllowHideDisplay</u>        | Indicates whether the end-user can hide the display at run time.                                |
| <u>Author</u>                  | Contains the author of the multimedia stream.   |
| <u>AutoRewind</u>              | Indicates whether to automatically rewind the multimedia stream when it stops.                  |
| <u>AutoStart</u>               | Indicates whether to automatically start playing the multimedia stream.                         |
| <u>Balance</u>                 | Specifies the stereo balance.   |
| <u>Copyright</u>               | Contains copyright information for this multimedia stream.                                      |
| <u>CurrentPosition</u>         | Specifies the current position within the multimedia stream, in seconds.                        |
| <u>CurrentState</u>            | Specifies the current state of the player: stopped, paused, running.                            |
| <u>Description</u>             | Contains a description for this multimedia stream.  |
| <u>DisplayBackColor</u>        | Specifies the color used for the control background.  |
| <u>DisplayForeColor</u>        | Specifies the color used for the control foreground.  |
| <u>DisplayMode</u>             | Indicates whether the control displays the current position in time or frames.                  |
| <u>Duration</u>                | Specifies the duration of the multimedia stream in seconds.                                     |
| <u>EnableContextMenu</u>       | Indicates whether to enable the context menu on right click.                                    |
| <u>EnablePositionControls</u>  | Indicates whether to enable the position buttons in the control panel.                          |
| <u>EnableSelectionControls</u> | Indicates whether to enable the selection buttons in the control panel.                         |
| <u>EnableTracker</u>           | Indicates whether to enable the   |

|                                     |   |
|-------------------------------------|---|
| <u><b>FileName</b></u>              | tracker bar in the control panel.<br>Specifies the name of the file that contains the multimedia stream to be played. |
| <u><b>FilterGraph</b></u>           | Returns an interface pointer (IUnknown *) to the current filter graph object.   |
| <u><b>FilterGraphDispatch</b></u>   | Returns an interface pointer (IDispatch *) to the current filter graph object.  |
| <u><b>ImageSourceHeight</b></u>     | Specifies the authored height of the source image.  |
| <u><b>ImageSourceWidth</b></u>      | Specifies the authored width of the source image.   |
| <u><b>MovieWindowSetting</b></u>    | Selects the image window size and characteristics.  |
| <u><b>PlayCount</b></u>             | Specifies the number of times to play this multimedia stream.   |
| <u><b>Rate</b></u>                  | Specifies the playback rate for the stream.   |
| <u><b>SelectionEnd</b></u>          | Specifies the ending position in this multimedia stream, in seconds, relative to the beginning of the stream.         |
| <u><b>SelectionStart</b></u>        | Specifies the starting position in this multimedia stream, in seconds, relative to the beginning of the stream.       |
| <u><b>ShowControls</b></u>          | Indicates whether the control panel is visible.   |
| <u><b>ShowDisplay</b></u>           | Indicates whether the status display panel is visible.  |
| <u><b>ShowPositionControls</b></u>  | Indicates whether the position buttons are visible in the control panel.  |
| <u><b>ShowSelectionControls</b></u> | Indicates whether the selection buttons are visible in the control panel.   |
| <u><b>ShowTracker</b></u>           | Indicates whether the tracker bar is visible in the control panel.  |
| <u><b>Title</b></u>                 | Specifies the title of the multimedia stream.   |
| <u><b>Volume</b></u>                | Specifies the audio volume.   |

The **ActiveMovie** control also supports properties that are common to other controls: **Appearance**, **BorderStyle**, **DragIcon**, **DragMode**, **Enabled**, **Height**, **HelpContextID**, **hWnd**, **Index**, **Left**, **Name**, **Parent**, **TabStop**, **TabIndex**, **Tag**, **Top**, **Visible**, **WhatsThisHelpID**, and **Width**. For information about these properties, please see your Visual Basic documentation.

**See Also**

**[ActiveMovie Control](#)**

**[ActiveMovie Methods](#)**

**[ActiveMovie Events](#)**

## ActiveMovie Methods

See Also

The **ActiveMovie** control supports the following methods:

| <b>Method</b>       | <b>Description</b>   |
|---------------------|--|
| <u><b>Pause</b></u> | Pause playing and maintain the current position in the multimedia stream.  |
| <u><b>Run</b></u>   | Play the multimedia stream.  |
| <u><b>Stop</b></u>  | Stop playback and reset the position as indicated by the <u>AutoRewind</u> and <u>SelectionStart</u> properties. |

The following methods are planned for future releases, but are not implemented in the current release:

| <b>Method</b>      | <b>Description</b>     |
|--------------------|------------------------|
| <b>FastForward</b> | (Not yet implemented.) |
| <b>Rewind</b>      | (Not yet implemented.) |
| <b>Seek</b>        | (Not yet implemented.) |

The **ActiveMovie** control also supports several methods that are common to other controls: **Drag**, **Move**, **SetFocus**, **ShowWhatsThis**, and **ZOrder**. For information about these methods, please see your Visual Basic documentation.

**See Also**

**ActiveMovie Control**

**ActiveMovie Properties**

**ActiveMovie Events**



## ActiveMovie Events

See Also

The **ActiveMovie** control supports the following events:

| <b>Event</b>                 | <b>Description</b>   |
|------------------------------|--|
| <u><b>Error</b></u>          | Indicates an error. (Note: Not yet implemented.)   |
| <u><b>PositionChange</b></u> | Indicates changes to the position, such as by the user seeking to the position using the default user interface. |
| <u><b>StateChange</b></u>    | Indicates player state changes, such as a change from stopped to running, or from running to paused.             |
| <u><b>Timer</b></u>          | Handles timer events.  |

The **ActiveMovie** control also supports several events that are common to other controls: **DragDrop**, **DragOver**, **GotFocus**, **KeyDown**, **KeyPress**, **KeyUp**, **LostFocus**, **MouseDown**, **MouseMove**, and **MouseUp**. For information about these events, please see your Visual Basic documentation.

**See Also**

**[ActiveMovie Control](#)**

**[ActiveMovie Properties](#)**

**[ActiveMovie Methods](#)**

# AllowChangeDisplayMode Property

## Applies To

ActiveMovie control

## Description

Indicates whether the end-user can change the **DisplayMode** property at run time.

## Syntax (Visual Basic)

[*form.*] *object* .**AllowChangeDisplayMode** [ = { **True** | **False** } ]

The syntax for the **AllowChangeDisplayMode** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

## Settings

The settings for **AllowChangeDisplayMode** are:

| Setting      | Description   |
|--------------|---|
| <b>True</b>  | (Default) Allow the user to change the <b>DisplayMode</b> property at run time. |
| <b>False</b> | Do not allow the user to change the <b>DisplayMode</b> property at run time.    |

## Type

Boolean

## Remarks

The **DisplayMode** property selects whether to show the current position of the multimedia stream in time or frames.

Run-time access: read-only. Design-time access: read/write.

## See Also

DisplayMode property

# AllowHideControls Property

## Applies To

ActiveMovie control

## Description

Indicates whether the end-user can hide the control panel at run time.

## Syntax (Visual Basic)

[*form.*] *object* .**AllowHideControls** [ = { **True** | **False** } ]

The syntax for the **AllowHideControls** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

## Settings

The settings for **AllowHideControls** are:

| Setting      | Description   |
|--------------|---|
| <b>True</b>  | (Default) Allow the user to change the <b>ShowControls</b> property to <b>False</b> . |
| <b>False</b> | Do not allow the user to change the <b>ShowControls</b> property to <b>False</b> .    |

## Type

Boolean

## Remarks

This property can be set only at design time. It then determines whether the user or the application can change the **ShowControls** property at run time.

Run-time access: read-only. Design-time access: read/write.

## See Also

ShowControls Property, AllowHideDisplay Property

# AllowHideDisplay Property

## Applies To

ActiveMovie control

## Description

Indicates whether the end-user can hide the status display panel at run time.

## Syntax (Visual Basic)

[*form.*] *object* .**AllowHideDisplay** [ = { **True** | **False** } ]

The syntax for the **AllowHideDisplay** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

## Settings

The settings for **AllowHideDisplay** are:

| Setting      | Description   |
|--------------|---|
| <b>True</b>  | (Default) Allow the user to change the <b>ShowDisplay</b> property at run time. |
| <b>False</b> | Do not allow the user to change the <b>ShowDisplay</b> property at run time.    |

## Type

Boolean

## Remarks

This property can be set only at design time. It then determines whether the user or an application can set the **ShowDisplay** property at run time.

Run-time access: read-only. Design-time access: read/write.

## See Also

ShowDisplay Property, AllowHideControls Property

# Author Property

## Applies To

ActiveMovie control

## Description

Contains the author of the multimedia stream.

## Syntax (Visual Basic)

[*form*.] *object* .**Author**

The syntax for the **Author** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

## Type

String

## Remarks

This value can be set at design time. When this value is present in the multimedia stream, it overwrites the value set at design time.

Run-time access: read-only. Design-time access: read/write.

## See Also

Copyright Property

# AutoRewind Property

## Applies To

ActiveMovie control

## Description

Indicates whether to automatically rewind the multimedia stream and reposition at the beginning after playing stops.

## Syntax (Visual Basic)

[*form*.] *object* .AutoRewind [ = { **True** | **False** } ]

The syntax for the **AutoRewind** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

## Settings

The settings for **AutoRewind** are:

| Setting      | Description   |
|--------------|---|
| <b>True</b>  | Reposition the multimedia stream at the beginning after playing.                  |
| <b>False</b> | (Default) Do not reposition the multimedia stream at the beginning after playing. |

## Type

Boolean

## Remarks

The rewind operation occurs after a stop operation and when a play operation reaches the position specified by the **SelectionEnd** property. The rewind operation resets the position to the value specified by the **SelectionStart** property.

To retain the current position within the multimedia stream, set **AutoRewind** to **False** or use the **Pause** method.

Run-time access: read/write. Design-time access: read/write.

## See Also

AutoStart Property, PlayCount Property

# AutoStart Property

## Applies To

ActiveMovie control

## Description

Indicates whether to automatically start playing the multimedia stream.

## Syntax (Visual Basic)

[*form*.] *object* .**AutoStart** [ = { **True** | **False** } ]

The syntax for the **AutoStart** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

## Settings

The settings for **AutoStart** are:

| Setting      | Description  |
|--------------|--|
| <b>True</b>  | Automatically start playing the multimedia stream.   |
| <b>False</b> | (Default) Do not automatically start the multimedia stream; require an explicit <b>Run</b> method. |

## Type

Boolean

## Remarks

Run-time access: read-only. Design-time access: read/write.

## See Also

PlayCount Property



# Balance Property

## Applies To

ActiveMovie control

## Description

Specifies the stereo balance.

## Syntax (Visual Basic)

[*form*.] *object* .**Balance** [ = *long* ]

The syntax for the **Balance** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

*long* A numeric expression that specifies the balance value. The number ranges from -10000 to +10000.

## Type

Long

## Remarks

The value 0 (the default value) indicates a neutral balance (no attenuation).

Run-time access: read/write. Design-time access: read/write.

## See Also

Volume Property

# Copyright Property

## Applies To

ActiveMovie control

## Description

Contains copyright information for this multimedia stream.

## Syntax (Visual Basic)

[*form*.] *object* .**Copyright**

The syntax for the **Copyright** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

## Type

String

## Remarks

This value can be set at design time. When this value is present in the multimedia stream, it overwrites the value set at design time.

Run-time access: read-only. Design-time access: read/write.

## See Also

Author Property, Title Property

## CurrentPosition Property

### Applies To

**ActiveMovie** control

### Description

Specifies the current position within the multimedia stream, in seconds.

### Syntax (Visual Basic)

[*form*.] *object* .**CurrentPosition** [ = *double* ]

The syntax for the **CurrentPosition** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

*double* Contains the new position within the stream, in seconds.

### Type

Double

### Remarks

The new value must be within the range specified by **SelectionStart** and **SelectionEnd**.

The current position value displayed by the control's user interface can represent either seconds or frames. The **DisplayMode** property determines the units shown.

Setting the **CurrentPosition** property at run time is similar to a seek operation and changes the position to the specified point in the multimedia stream.

Run-time access: read/write. Design-time access: not applicable.

### See Also

**DisplayMode** Property, **SelectionStart** Property, **SelectionEnd** Property

## CurrentState Property

### Applies To

ActiveMovie control

### Description

Contains the active state of the control, such as running, paused, or stopped.

### Syntax (Visual Basic)

[*form.*] *object* .**CurrentState**

The syntax for the **CurrentState** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

### Settings

The settings for **CurrentState** are:

| <b>Setting</b>    | <b>Value</b> | <b>Description</b>  |
|-------------------|--------------|---|
| <b>amvStopped</b> | 0            | The player is stopped.  |
| <b>amvPaused</b>  | 1            | The player is paused.   |
| <b>amvRunning</b> | 2            | The player is running and actively playing the multimedia stream. |

### Type

Long

### Remarks

To change the state value, call one of the **ActiveMovie** control methods, such as **Run**, **Pause**, or **Stop**.

Run-time access: read-only. Design-time access: not applicable.

### See Also

Run method, Pause method, Stop method

## Description Property

### Applies To

**ActiveMovie** control

### Description

Contains a description for this multimedia stream.

### Syntax (Visual Basic)

*[form.] object* .**Description**

The syntax for the **Description** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

### Type

String

### Remarks

This value can be set at design time. When this value is present in the multimedia stream, it overwrites the value set at design time.

Run-time access: read-only. Design-time access: read/write.

### See Also

**Author** Property

# DisplayBackColor Property

## Applies To

ActiveMovie control

## Description

Specifies the color used for the display panel background.

## Syntax (Visual Basic)

[*form*.] *object* .DisplayBackColor [ = *color* ]

The syntax for the **DisplayBackColor** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

*color* A value or constant that determines the background, as described in Settings.

## Settings

The settings for color are:

| Setting               | Description  |
|-----------------------|--|
| Normal RGB colors     | Colors specified by using the <b>Color</b> palette or by using the <b>RGB</b> or <b>QBColor</b> functions in code.   |
| System default colors | Colors specified by system color constants listed in the object library in the <b>Object Browser</b> . The Windows operating environment substitutes the user's choices as specified in the <b>Control Panel</b> settings. |

## Remarks

At design time, the default setting is the system default color specified by the constant **vbWindowBackground**.

The valid range for a normal RGB color is 0 to 16,777,215 (&HFFFFFF). The high byte of a number in this range equals 0; the lower 3 bytes, from least to most significant byte, determine the amount of red, green, and blue, respectively. The red, green, and blue components are each represented by a number between 0 and 255 (&HFF). If the high byte isn't 0, Visual Basic uses the system colors, as defined in the user's **Control Panel** settings and by constants listed in the object library in the **Object Browser**.

Run-time access: read/write. Design-time access: read/write.

## See Also

DisplayForeColor Property

# DisplayForeColor Property

## Applies To

ActiveMovie control

## Description

Specifies the color used for the display panel foreground.

## Syntax (Visual Basic)

[*form*.] *object* .DisplayForeColor [ = *color* ]

The syntax for the **DisplayForeColor** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

*color* A value or constant that determines the background, as described in Settings.

## Settings

The settings for color are:

| Setting               | Description  |
|-----------------------|--|
| Normal RGB colors     | Colors specified by using the <b>Color</b> palette or by using the <b>RGB</b> or <b>QBColor</b> functions in code.   |
| System default colors | Colors specified by system color constants listed in the object library in the <b>Object Browser</b> . The Windows operating environment substitutes the user's choices as specified in the <b>Control Panel</b> settings. |

## Remarks

At design time, the default setting is the system default color specified by the constant **vbWindowText**.

The valid range for a normal RGB color is 0 to 16,777,215 (&HFFFFFF). The high byte of a number in this range equals 0; the lower 3 bytes, from least to most significant byte, determine the amount of red, green, and blue, respectively. The red, green, and blue components are each represented by a number between 0 and 255 (&HFF). If the high byte isn't 0, Visual Basic uses the system colors, as defined in the user's **Control Panel** settings and by constants listed in the object library in the **Object Browser**.

Run-time access: read/write. Design-time access: read/write.

## See Also

DisplayBackColor Property

# DisplayMode Property

## Applies To

**ActiveMovie** control

## Description

Indicates the units for the current position value when the position value is displayed.

## Syntax (Visual Basic)

*[form.] object .DisplayMode* [= *setting* ]

The syntax for the **DisplayMode** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

*setting* Specifies the new value, as described in Settings, below.

## Settings

The settings for **DisplayMode** are:

| <b>Setting</b>   | <b>Value</b> | <b>Description</b>                                 |
|------------------|--------------|--|
| <b>amvTime</b>   | 0            | (Default) Display the current position in seconds. |
| <b>amvFrames</b> | 1            | Display the current position in frames.            |

## Type

Integer

## Remarks

The properties **ShowDisplay** and **AllowHideDisplay** determine whether the display panel appears on the **ActiveMovie** control.

Run-time access: read/write. Design-time access: read/write.

## See Also

**AllowHideDisplay** Property, **ShowDisplay** Property



# Duration Property

## Applies To

**ActiveMovie** control

## Description

Specifies the duration of the multimedia stream in seconds.

## Syntax (Visual Basic)

*[form.] object* .**Duration**

The syntax for the **Duration** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

## Type

Double

## Remarks

The multimedia stream is contained in the file specified by the **FileName** property. The **Duration** property always represents the length of the entire stream, not just the part of the stream indicated by the **SelectionStart** and **SelectionEnd** properties.

Run-time access: read-only. Design-time access: not applicable.

## See Also

**FileName** Property

# EnableContextMenu Property

## Applies To

ActiveMovie control

## Description

Indicates whether to enable the context menu on right click.

## Syntax (Visual Basic)

[*form.*] *object* .EnableContextMenu [ = { **True** | **False** } ]

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

## Settings

The settings for **EnableContextMenu** are:

| Setting      | Description                                       |
|--------------|---|
| <b>True</b>  | (Default) Enable the context menu on right click. |
| <b>False</b> | Do not enable the context menu on right click.    |

## Type

Boolean

## Remarks

Run-time access: read/write. Design-time access: read/write.

## See Also

ShowDisplay Property, ShowControls Property

## EnablePositionControls Property

### Applies To

ActiveMovie control

### Description

Indicates whether to enable the position controls.

### Syntax (Visual Basic)

[*form*.] *object* .EnablePositionControls [ = { **True** | **False** } ]

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

### Settings

The settings for **EnablePositionControls** are:

| Setting      | Description                             |
|--------------|---|
| <b>True</b>  | (Default) Enable the position controls. |
| <b>False</b> | Disable (dim) the position controls.    |

### Type

Boolean

### Remarks

Run-time access: read/write. Design-time access: read/write.

### See Also

ShowPositionControls Property

## EnableSelectionControls Property

### Applies To

ActiveMovie control

### Description

Indicates whether to enable the position controls.

### Syntax (Visual Basic)

[*form*.] *object* .EnableSelectionControls [ = { True | False } ]

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

### Settings

The settings for **EnableSelectionControls** are:

| Setting | Description                              |
|---------|--|
| True    | (Default) Enable the selection controls. |
| False   | Disable (dim) the selection controls.    |

### Type

Boolean

### Remarks

Run-time access: read/write. Design-time access: read/write.

### See Also

ShowSelectionControls Property

# EnableTracker Property

## Applies To

ActiveMovie control

## Description

Indicates whether to enable the tracker.

## Syntax (Visual Basic)

[*form.*] *object* .**EnableTracker** [ = { **True** | **False** } ]

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

## Settings

The settings for **EnableTracker** are:

| <b>Setting</b> | <b>Description</b>            |
|----------------|-------------------------------|
| <b>True</b>    | (Default) Enable the tracker. |
| <b>False</b>   | Disable (dim) the tracker.    |

## Type

Boolean

## Remarks

Run-time access: read/write. Design-time access: read/write.

## See Also

ShowTracker Property

## FileName Property

### Applies To

**ActiveMovie** control

### Description

Specifies the name of the file that contains the multimedia stream to be played.

### Syntax (Visual Basic)

[*form*.] *object* .**FileName** [ = *string* ]

The syntax for the **FileName** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

*string* Contains the name of the file that contains the multimedia stream.

### Type

String

### Remarks

Run-time access: read/write. Design-time access: read/write.

### See Also

**Description** Property

## FilterGraph Property

### Applies To

ActiveMovie control

### Description

Contains an **IUnknown** interface pointer to the current filter graph object.

### Syntax (Visual Basic)

[*form.*] *object* .**FilterGraph** [ = *punk* ]

The syntax for the **FilterGraph** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

*punk* Contains the **IUnknown** pointer for the filter graph object, **IFilterGraph**.

### Type

IUnknown \*

### Remarks

The filter graph represents a specific configuration of source, transform, and rendering filters. The filter graph represents the complete set of software components needed to process a given multimedia stream within the ActiveMovie architecture.

You can set this property to change the current filter graph.

An **IDispatch** interface pointer for the filter graph object is available through the **FilterGraphDispatch** property.

Run-time access: read/write. Design-time access: read/write.

For more information about the **IFilterGraph** object, see the ActiveMovie SDK documentation.

### See Also

FilterGraphDispatch Property, FileName Property

## FilterGraphDispatch Property

### Applies To

[ActiveMovie control](#)

### Description

Contains an **IDispatch** interface pointer to the current filter graph object.

### Syntax (Visual Basic)

[*form.*] *object* .**FilterGraphDispatch** [ = *pdisp* ]

The syntax for the **FilterGraphDispatch** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

*pdisp* Contains the **IDispatch** pointer to the current filter graph object.

### Type

IDispatch \*

### Remarks

The filter graph represents a specific configuration of source, transform, and rendering filters. The filter graph represents the complete set of software components needed to process a given multimedia stream within the ActiveMovie architecture.

You can set this property to change the current filter graph.

An **IUnknown** interface pointer for the filter graph object is available through the **FilterGraph** property.

For more information about the **IFilterGraph** object, see the ActiveMovie SDK documentation.

Run-time access: read-only. Design-time access: read/write.

### See Also

[FilterGraph Property](#), [FileName Property](#)



## ImageSourceHeight Property

### Applies To

**ActiveMovie** control

### Description

Specifies the authored height of the source image.

### Syntax (Visual Basic)

*[form.] object* .**ImageSourceHeight**

The syntax for the **ImageSourceHeight** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

### Type

Long

### Remarks

This value is independent of the projected image size, which is determined by the **MovieWindowSetting** property.

Run-time access: read-only. Design-time access: read/write.

### See Also

**MovieWindowSetting** property, **ImageSourceWidth** Property

## ImageSourceWidth Property

### Applies To

ActiveMovie control

### Description

Specifies the authored width of the source image.

### Syntax (Visual Basic)

*[form.] object* .ImageSourceWidth

The syntax for the **ImageSourceWidth** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

### Type

Long

### Remarks

This value is independent of the projected image size, which is determined by the **MovieWindowSetting** property.

Run-time access: read-only. Design-time access: read/write.

### See Also

MovieWindowSetting Property, ImageSourceHeight Property

# MovieWindowSetting Property

## Applies To

ActiveMovie control

## Description

Specifies the settings for the window that displays the images associated with a multimedia stream.

## Syntax (Visual Basic)

[*form.*] *object* .**MovieWindowSetting** [ = *setting* ]

The syntax for the **MovieWindowSetting** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

*setting* Specifies the window setting, described in Settings, below.

## Settings

The settings for **MovieWindowSetting** are:

| Setting                          | Value | Description   |
|----------------------------------|-------|---|
| <b>amvDefaultSize</b>            | 0     | (default) Uses the authored size.   |
| <b>amvHalfSize</b>               | 1     | Reduces the image projection size to exactly half the authored size.  |
| <b>amvDoubleSize</b>             | 2     | Increases the image projection size to twice the authored size.   |
| <b>amvMaximized</b>              | 3     | Maximizes the image size within its parent form.  |
| <b>amvFullScreen</b>             | 4     | Projects the images onto the full screen.   |
| <b>amvPermitResizeWithAspect</b> | 5     | Allows users to resize the form that contains the <b>ActiveMovie</b> control, while retaining the authored aspect ratio.                  |
| <b>amvPermitResizeNoRestrict</b> | 6     | Allows users to resize the form that contains the <b>ActiveMovie</b> control, fitting the image to the resized form without restrictions. |

## Type

Long

## Remarks

Run-time access: read/write. Design-time access: read/write.

## See Also

ImageSourceHeight Property

# PlayCount Property

## Applies To

[ActiveMovie control](#)

## Description

Specifies the number of times to play the multimedia stream.

## Syntax (Visual Basic)

*[form.] object .PlayCount [ = long ]*

The syntax for the **PlayCount** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

*long* A numeric expression that specifies the number of times to play the multimedia stream. The value 0 indicates play repeatedly.

## Type

Long

## Remarks

The value 0 indicates that the control should play the multimedia stream repeatedly, restarting as soon as it finishes playing the stream.

Run-time access: read/write. Design-time access: read/write.

## See Also

[AutoStart Property](#)

# Rate Property

## Applies To

[ActiveMovie control](#)

## Description

Specifies the playback rate for the multimedia stream.

## Syntax (Visual Basic)

[*form*.] *object* .**Rate** [ = *double* ]

The syntax for the **Rate** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

*double* A numeric expression that represents the playback rate, where 1.0 corresponds to the authored rate.

## Type

Double

## Remarks

This acts as a multiplier value that allows the stream to be played in slow motion or in fast motion. The value 1.0 indicates normal, or authored, speed. Note that the audio track becomes difficult to understand at rates lower than 0.5 and higher than 1.5.

The default value is 1.0.

Run-time access: read/write. Design-time access: read/write.

## See Also

[Duration Property](#)

# Rating Property

## Applies To

ActiveMovie control

## Description

Contains rating information relating to the multimedia stream.

## Syntax (Visual Basic)

*[form.] object* .**Rating**

The syntax for the **Rating** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

## Type

String

## Remarks

This value can be set at design time. When this value is present in the multimedia stream, it overwrites the value set at design time.

Run-time access: read-only. Design-time access: read/write.

## See Also

Description Property

## SelectionEnd Property

### Applies To

ActiveMovie control

### Description

Specifies the ending position in this multimedia stream, in seconds, relative to the beginning of the stream.

### Syntax (Visual Basic)

[*form*.] *object* .**SelectionEnd** [ = *double* ]

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

*double* A numeric expression that specifies the position within the multimedia stream that represents the end of the playback sequence.

### Type

Double

### Remarks

The default value for the **SelectionEnd** is the **Duration** property.

Run-time access: read/write. Design-time access: read/write.

### See Also

Duration Property, SelectionStart Property



## SelectionStart Property

### Applies To

ActiveMovie control

### Description

Specifies the starting position in this multimedia stream, in seconds, relative to the beginning of the stream.

### Syntax (Visual Basic)

[*form*.] *object* .**SelectionStart** [ = *double* ]

The syntax for the **SelectionStart** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

*double* A numeric expression that specifies the position within the multimedia stream that represents the beginning of the playback sequence.

### Type

Double

### Remarks

The default value for **SelectionStart** is 0.

Run-time access: read/write. Design-time access: read/write.

### See Also

Duration Property, SelectionEnd Property

# ShowControls Property

## Applies To

**ActiveMovie** control

## Description

Indicates whether the control panel is visible.

## Syntax (Visual Basic)

[*form.*] *object* .**ShowControls** [ = { **True** | **False** } ]

The syntax for the **ShowControls** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

## Settings

The settings for **ShowControls** are:

| <b>Setting</b> | <b>Description</b>                            |
|----------------|---|
| <b>True</b>    | (Default) Show the control panel at run time. |
| <b>False</b>   | Do not show the control panel at run time.    |

## Type

Boolean

## Remarks

Run-time access: read/write. Design-time access: read/write.

## See Also

**ShowDisplay** Property

# ShowDisplay Property

## Applies To

ActiveMovie control

## Description

Indicates whether the status display panel is visible.

## Syntax (Visual Basic)

[*form*.] *object* .**ShowDisplay** [ = { **True** | **False** } ]

The syntax for the **ShowDisplay** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

## Settings

The settings for **ShowDisplay** are:

| <b>Setting</b> | <b>Description</b>                             |
|----------------|--|
| <b>True</b>    | (Default) Show the status display at run time. |
| <b>False</b>   | Do not show the status display at run time.    |

## Type

Boolean

## Remarks

Run-time access: read/write. Design-time access: read/write.

## See Also

ShowControls Property

# ShowPositionControls Property

## Applies To

ActiveMovie control

## Description

Indicates whether the position controls are visible.

## Syntax (Visual Basic)

[*form.*] *object* .**ShowPositionControls** [ = { **True** | **False** } ]

The syntax for the **ShowPositionControls** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

## Settings

The settings for **ShowPositionControls** are:

| <b>Setting</b> | <b>Description</b>                                       |
|----------------|--|
| <b>True</b>    | Show the position controls at run time.                  |
| <b>False</b>   | (Default) Do not show the position controls at run time. |

## Type

Boolean

## Remarks

Run-time access: read/write. Design-time access: read/write.

## See Also

EnablePositionControls Property

# ShowSelectionControls Property

## Applies To

**ActiveMovie** control

## Description

Indicates whether the selection controls are visible.

## Syntax (Visual Basic)

[*form*.] *object* .**ShowSelectionControls** [ = { **True** | **False** } ]

The syntax for the **ShowSelectionControls** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

## Settings

The settings for **ShowSelectionControls** are:

| <b>Setting</b> | <b>Description</b>  |
|----------------|---|
| <b>True</b>    | Show the selection controls at run time.                  |
| <b>False</b>   | (Default) Do not show the selection controls at run time. |

## Type

Boolean

## Remarks

Run-time access: read/write. Design-time access: read/write.

## See Also

**EnableSelectionControls** Property

# ShowTracker Property

## Applies To

ActiveMovie control

## Description

Indicates whether the tracker is visible.

## Syntax (Visual Basic)

[*form*.] *object* .**ShowTracker** [ = { **True** | **False** } ]

The syntax for the **ShowTracker** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

## Settings

The settings for **ShowTracker** are:

| <b>Setting</b> | <b>Description</b>                      |
|----------------|---|
| <b>True</b>    | (Default) Show the tracker at run time. |
| <b>False</b>   | Do not show the tracker at run time.    |

## Type

Boolean

## Remarks

Run-time access: read/write. Design-time access: read/write.

## See Also

EnableTracker Property

## Title Property (ActiveMovie Control)

### Applies To

ActiveMovie control

### Description

Specifies the title of the multimedia stream.

### Syntax (Visual Basic)

*[form.] object .Title*

The syntax for the **Title** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

### Type

String

### Remarks

This value can be set at design time. When this value is present in the multimedia stream, it overwrites the value set at design time.

Run-time access: read-only. Design-time access: read/write.

### See Also

Author Property

# Volume Property

## Applies To

ActiveMovie control

## Description

Specifies the volume, in hundredths of decibels.

## Syntax (Visual Basic)

[*form*.] *object* .**Volume** [ = *long* ]

The syntax for the **Volume** property has these parts:

*form* An object expression that evaluates to a Visual Basic form.

*object* An object expression that evaluates to an **ActiveMovie** control.

*long* A numeric expression that specifies the audio volume, in hundredths of decibels.

## Type

Long

## Remarks

The value ranges from **AX\_MIN\_VOLUME**, -10000, to **AX\_MAX\_VOLUME**, 0. The value 0 (the default value) represents full volume.

Run-time access: read/write. Design-time access: read/write.

## See Also

Balance Property



# Pause Method

## Applies To

[ActiveMovie control](#)

## Description

Suspends a play operation without changing the current position.

## Syntax (Visual Basic)

*object* .**Pause**

## Parameters

*object* An object expression that evaluates to an **ActiveMovie** control.

## Remarks

The **Pause** method pauses the multimedia stream at the current position. To continue playing the multimedia stream, use the **Run** method.

To stop the multimedia stream, use the **Stop** method.

## See Also

[Stop Method](#), [Run Method](#)

## Run Method

### Applies To

**ActiveMovie** control

### Description

Starts a multimedia stream from the specified starting position or continues playing a paused stream.

### Syntax (Visual Basic)

*object* .Run

### Parameters

*object* An object expression that evaluates to an **ActiveMovie** control.

### Remarks

The **Run** method starts the multimedia stream at the starting position specified by the **SelectionStart** property. In the absence of other user or application input, the **Run** method continues playing the stream to the position specified by the **SelectionEnd** property.

The **Run** method is also used to resume playing a paused multimedia stream.

To pause playing, call the **Pause** method. To stop playing, call the **Stop** method.

### See Also

**SelectionStart** Property, **Pause** Method, **Stop** Method

## Stop Method (ActiveMovie Control)

### Applies To

ActiveMovie control

### Description

Stops the playing of a multimedia stream.

### Syntax (Visual Basic)

*object* .**Stop**

### Parameters

*object* An object expression that evaluates to an **ActiveMovie** control.

### Remarks

The **Stop** method changes the **CurrentState** property.

When the **Stop** method halts a play operation, the current position can be optionally reset to the starting point for the multimedia stream, as indicated by the **SelectionStart** property. The **AutoRewind** property determines whether the position is reset to this starting position.

To halt a play operation without changing the current position, use the **Pause** method.

### See Also

AutoRewind Property, Run Method, SelectionStart Property

## Error Event (ActiveMovie Control)

### Applies To

ActiveMovie control

### Description

Raises an event when an error occurs.

### Syntax (Visual Basic)

```
Private Sub object_Error(ByVal SCode As Integer, ByVal Description As String, ByVal Source As String, CancelDisplay As Boolean)
```

### Parameters

*object* An object expression that evaluates to an **ActiveMovie** control.

*SCode* An error code.

*Description* A string describing the error which occurred.

*Source* A string containing the ActiveMovie Control's name.

*Cancel Display* A value that may be set by the client to cancel the default error messages.

### Remarks

The **Error** event is fired when **ActiveMovie** reports an error during playback. By default, the **ActiveMovie** control displays a message box containing the description string. To avoid displaying this box, set the **CancelDisplay** parameter of the **Error** event to False.

## PositionChange Event

### Applies To

ActiveMovie control

### Description

Indicates changes to the position that are not otherwise accessible to the client.

### Syntax (Visual Basic)

**Private Sub *object*\_PositionChange(ByVal *oldPosition* As Double, ByVal *newPosition* As Double)**

### Parameters

*object* An object expression that evaluates to an **ActiveMovie** control.

*oldPosition* The position before it was changed in seconds.

*newPosition* The current position in seconds, after the position change occurred.

### Remarks

The **PositionChange** event is raised to indicate changes to the position that are not otherwise accessible to the client. For example, the event is raised when the default UI is used to seek to a position within the multimedia stream.

Changes made directly to the **CurrentPosition** property do not trigger this event.

## StateChange Event

### Applies To

ActiveMovie control

### Description

Indicates changes to the player state, such as from stopped to running.

### Syntax (Visual Basic)

**Private Sub *object*\_StateChange(ByVal *oldState* As Long, ByVal *newState* As Long)**

### Parameters

*object* An object expression that evaluates to an **ActiveMovie** control.

*oldState* The previous state, before the change occurred.

*newState* The current state, after the change occurred.

### Remarks

The **StateChange** event is raised when the player state changes, such as from stopped to running. The current player state appears in the **CurrentState** property. State flags are listed under **CurrentState** property.

## Timer Event

### Applies To

ActiveMovie control

### Description

Raised by the **ActiveMovie** control's internal timer.

### Syntax (Visual Basic)

**Private Sub** *object\_Timer*( )

### Parameters

*object* An object expression that evaluates to an **ActiveMovie** control.

### Remarks

The **Timer** event is raised at the intervals specified by the control's timer.

## ActiveMovie Control Shortcut Keys

### ActiveMovie Control

The following key combinations can be used to activate **ActiveMovie** commands:

| <b>Key(s)</b>          | <b>Result</b>                                 |
|------------------------|---|
| CTRL+D                 | Toggle display                                |
| CTRL+ENTER             | Toggle between full screen and windowed modes |
| CTRL+LEFT ARROW        | Rewind  |
| CTRL+P                 | Pause   |
| CTRL+R                 | Run   |
| CTRL+RIGHT ARROW       | Forward                                       |
| CTRL+S                 | Stop  |
| CTRL+SHIFT+LEFT ARROW  | Previous                                      |
| CTRL+SHIFT+RIGHT ARROW | Next  |
| CTRL+T                 | Toggle control panel                          |



## ActiveMovie Property Pages

A property page is a tabbed dialog box that you can use to set properties on a control. You can access the ActiveMovie property page at either design or run-time.

To activate a property page at design time, select the control, and press F4, and then, in the Properties window, click on the Custom button.

To activate a property page at run-time, click on your right mouse button, and then click Properties on the popup menu that appears.

[Playback](#)

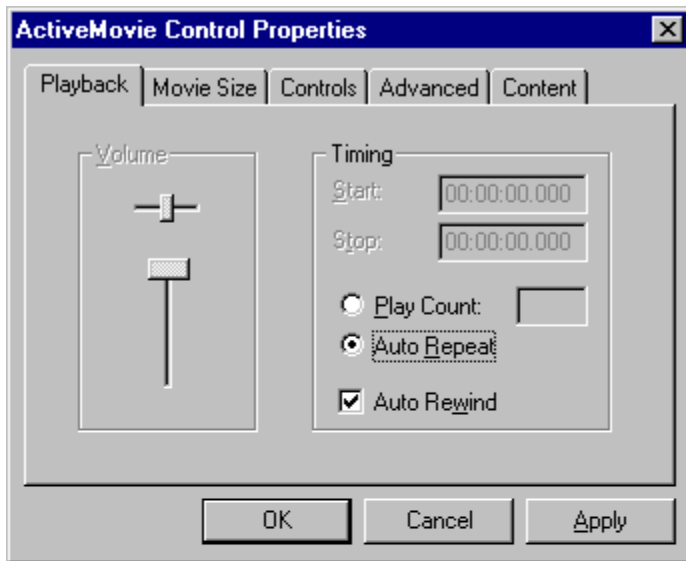
[Movie Size](#)

[Controls](#)

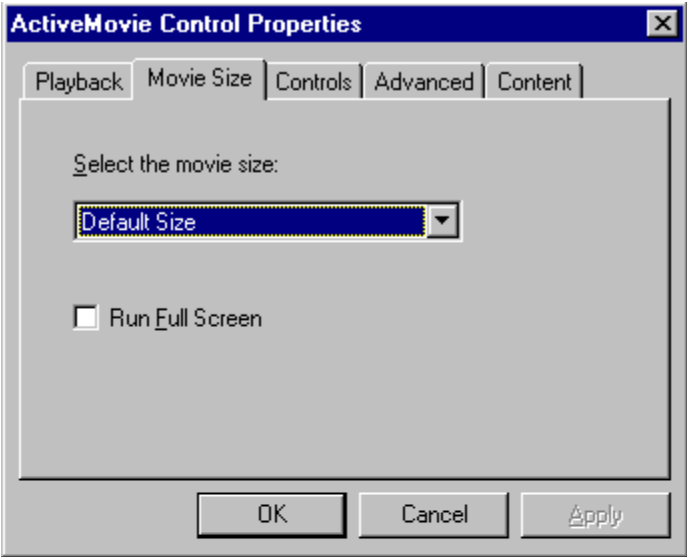
[Advanced](#)

[Content](#)

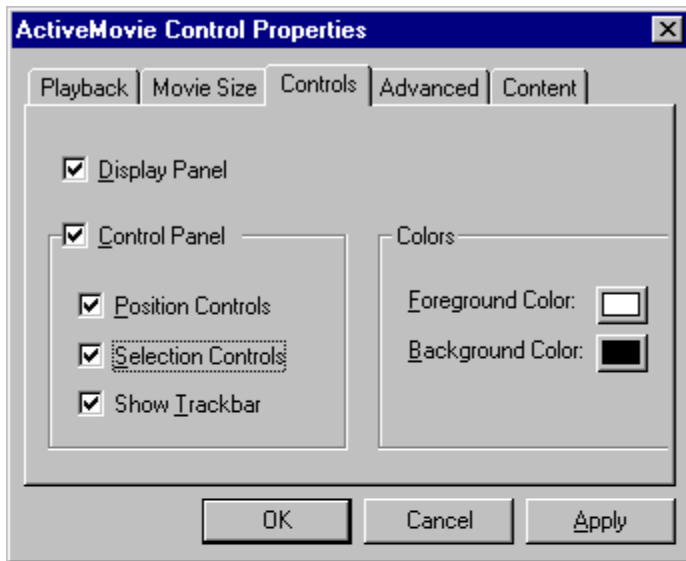
# Playback



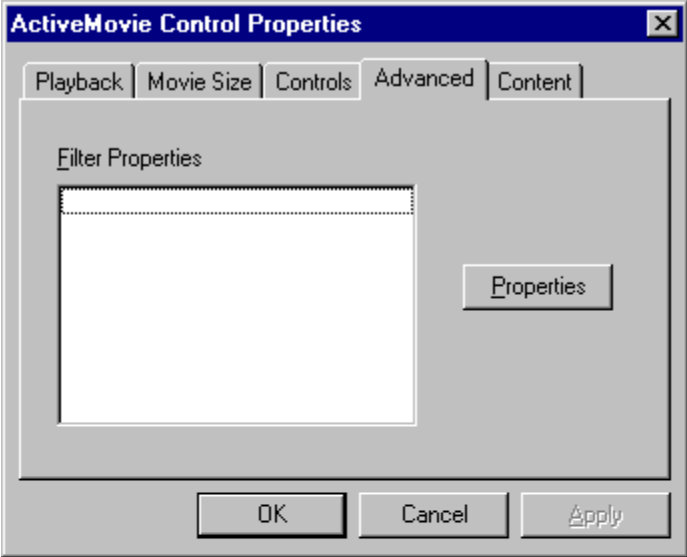
# Movie Size



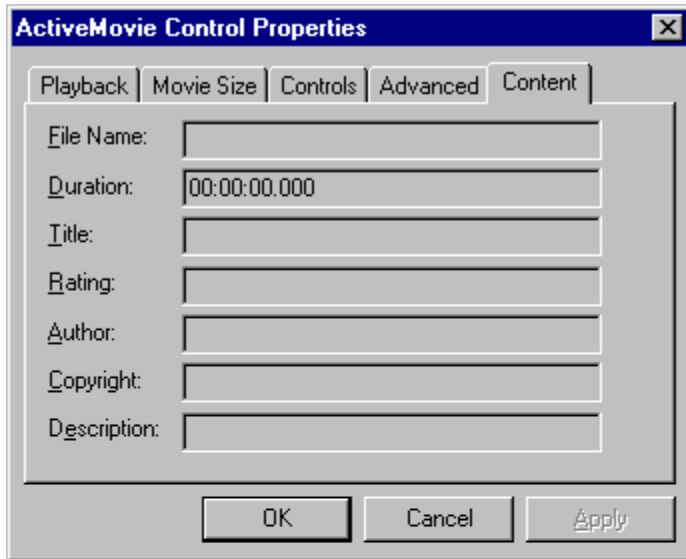
## Controls



# Advanced



# Content



The image shows a dialog box titled "ActiveMovie Control Properties" with a close button (X) in the top right corner. The dialog has five tabs: "Playback", "Movie Size", "Controls", "Advanced", and "Content". The "Content" tab is currently selected. Inside the dialog, there are seven text input fields, each with a label to its left: "File Name:", "Duration:", "Title:", "Rating:", "Author:", "Copyright:", and "Description:". The "Duration:" field contains the text "00:00:00.000". At the bottom of the dialog, there are three buttons: "OK", "Cancel", and "Apply".

| Field Label  | Value        |
|--------------|--------------|
| File Name:   |              |
| Duration:    | 00:00:00.000 |
| Title:       |              |
| Rating:      |              |
| Author:      |              |
| Copyright:   |              |
| Description: |              |

Buttons: OK, Cancel, Apply

## **Volume and Balance**

Toggles the volume and balance for the OCX.

For information on how to set this property in code, see [Volume](#) and [Balance](#).

## Start and Stop Time

Identifies the start and stop time for the current file.

For information on how to set this property in code, see [AllowChangeDisplayMode](#).



## Play Count

Specifies the number of times to play the current file consecutively.

For information on how to set this property in code, see [PlayCount](#).

## Auto Repeat

Causes the **ActiveMovie** control to repeatedly play the current file.

For information on how to set this property in code, see [AutoStart](#).

## Auto Rewind

Indicates whether to automatically rewind the multimedia stream and reposition at the beginning after playing stops.

For information on how to set this property in code, see [AutoRewind](#).

## Filename

Displays the name of the currently loaded file.

For information on how to set this property in code, see [FileName](#).

## Duration

Displays the duration of the currently loaded file.

For information on how to set this property in code, see [Duration](#).

## Title

Displays the title of the currently loaded file.

For information on how to set this property in code, see [Title](#).

## Rating

Displays the rating of the currently loaded file.

For information on how to set this property in code, see [Rating](#).

## Author

Identifies the author of the current file.

For information on how to set this property in code, see [Author](#).



## Copyright

Contains copyright information for the current file.

For information on how to set this property in code, see [Copyright](#).

## Description

Contains a description of the current file.

For information on how to set this property in code, see [Description](#).

## Display Panel

Check this box if you want the display panel to be visible at run-time.

For information on how to set this property in code, see [ShowDisplay](#).

## Control Panel

Check this box if you want the control panel to be visible at run-time.

For more information on how to set this property in code, see [ShowControls](#).

## Position Control

Check this box if you want the video stream position to be displayed.

For information on how to set this property in code, see [DisplayMode](#).

## Selection Controls

Check this box to make the selection controls visible.

For information on how to set this property in code, see [SelectionStart](#) and [SelectionEnd](#).

## Show Trackbar

Click this box to toggle the visibility of the track bar.

For information on how to set this property in code, see [ShowTracker](#).

## Background and Foreground Color

Use these controls to show and set the background and foreground colors of the ActiveMovie control.

For information on how to set this property in code, see [DisplayForeColor](#) and [DisplayBackColor](#).



## Run Full Screen

Choose this option to maximize the ActiveMovie control so that it fills the entire screen.

For information on how to set this property in code, see [MovieWindowSetting](#) .

## Movie Size

Use this option list to specify the movie's size.

For information on how to set this property in code, see [MovieWindowSetting](#).

## Run Full Screen

Choose this option if you want the ActiveMovie images projected onto the full screen.

For information on how to set this property in code, see [MovieWindowSetting](#).

## Filter Properties

Identifies the filters that you can set on the current video stream. For more details, see the [ActiveMovie SDK documentation](#).

## Properties

Applies the filter selected in the Filter Properties list. For more details, see the ActiveMovie SDK documentation.

## **Introduction**

This document describes WebBrowser, an ActiveX control that developers can use to add Internet browsing capabilities to their applications, and InternetExplorer, an OLE Automation object that developers can use to control the Microsoft Internet Explorer (IE) application from within an application.

## **WebBrowser Object**

The **WebBrowser** object is an ActiveX control that allows you to add browsing capabilities to your applications. The web browser control can be used to browse sites on the World Wide Web, as well as directories on the local machine and on network servers.

### **Properties**

Application, Busy, Container, Document, Height, Left, LocationName, LocationURL, Parent, Top, TopLevelContainer, Type, Width

### **Methods**

GoBack, GoForward, GoHome, GoSearch, Navigate, Refresh, Refresh2, Stop

### **Events**

OnBeginNavigate, OnCommandStateChange, OnDownloadBegin, OnDownloadComplete, OnNavigate, OnNewWindow, OnProgress, OnStatusTextChange

## InternetExplorer Object

The InternetExplorer object allows an application to create and control an instance of the Microsoft Internet Explorer application.

### Properties

Application, Busy, Container, Document, FullName, FullScreen, Height, HWND, Left, LocationName, LocationURL, MenuBar, Name, Parent, Path, StatusBar, StatusText, ToolBar, Top, TopLevelContainer, Type, Visible, Width

### Methods

ClientToWindow, GetProperty, GoBack, GoForward, GoHome, GoSearch, Navigate, PutProperty, Quit, Refresh, Refresh2, Stop

### Events

OnBeginNavigate, OnCommandStateChange, OnDownloadBegin, OnDownloadComplete, OnNavigate, OnNewWindow, OnProgress, OnPropertyChange, OnQuit, OnStatusTextChange, OnWindowActivated, OnWindowMove, OnWindowSized



## **Application Property (WebBrowser Object)**

*object*.Application

Returns the automation object supported by the application that contains the web browser control, if the object is accessible; otherwise this property returns the web browser control's automation object.

*object*

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

### **Applies To**

InternetExplorer, WebBrowser

## **Busy Property (WebBrowser Object)**

*object.Busy*

Returns a Boolean value specifying whether the web browser control or Internet Explorer is engaged in a downloading operation or other activity.

- The **Busy** property returns these values:

| <b>Value</b> | <b>Description</b>                             |
|--------------|--|
| <b>True</b>  | A download or other operation is in progress.  |
| <b>False</b> | No download or other operation is in progress. |

*object*

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

### **Applies To**

InternetExplorer, WebBrowser

## **Container Property (WebBrowser Object)**

*object.Container*

Returns an object that evaluates to the container of the web browser control, if any.

*object*

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

### **Applies To**

InternetExplorer, WebBrowser

## **Document Property (WebBrowser Object)**

*object.Document*

Returns the automation object of the active document, if any.

*object*

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

### **Applies To**

InternetExplorer, WebBrowser

## **FullName Property (InternetExplorer Object)**

*object.FullName*

Returns a string that evaluates to the fully qualified pathname of the executable file that contains the Internet Explorer application.

*object*

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

### **Applies To**

InternetExplorer

## **FullScreen Property (InternetExplorer Object)**

*object*.FullScreen [= *value*]

Returns or sets a value indicating whether Internet Explorer is in full screen or normal window mode. In full screen mode, Internet Explorer's main window is maximized and the status bar, tool bar, menu bar, and title bar are hidden.

*object*

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

*value*

Optional. A Boolean expression that determines whether Internet Explorer is in full screen or normal window mode. If **True**, the object is in full screen mode; if **False**, it is in normal mode.

### **Applies To**

InternetExplorer

## Height Property (WebBrowser Object)

*object.Height* [= *height*]

Returns or sets the vertical dimension, in pixels, of the frame window that contains the web browser control.

*object*

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

*height*

Optional. A long integer value specifying the vertical dimension of the frame window, in pixels.

### **Applies To**

InternetExplorer, WebBrowser

## **HWND Property (InternetExplorer Object)**

*object.HWND*

Returns the handle of Internet Explorer's main window.

*object*

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

### **Applies To**

InternetExplorer



## **Left Property (InternetExplorer Object)**

*object.Left* [= *distance*]

Returns or sets the distance between the internal left edge of the web browser control and the left edge of its container.

*object*

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

*distance*

Optional. A long integer expression specifying the distance between the internal left edge of the web browser control and the left edge of its container.

The Left property is measured in units depending on the coordinate system of its container. The values for this property changes as the object is moved by the user or by code.

### **Applies To**

InternetExplorer, WebBrowser

## **LocationName Property (WebBrowser Object)**

*object*.LocationName

Returns a string that contains the name of the resource that the web browser is currently displaying. If the resource is a HTML page on the World Wide Web, the name is the title of that page. If the resource is a folder or file on the network or local machine, the name is the UNC or full pathname of the folder or file.

*object*

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

### **Applies To**

InternetExplorer, WebBrowser

## **LocationURL Property**

*object*.LocationURL

Returns a string that contains the URL of the resource that the web browser control or Internet Explorer is currently displaying. If the resource is a folder or file on the network or local machine, the name is the UNC or full pathname of the folder or file.

*object*

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

### **Applies To**

InternetExplorer, WebBrowser

## MenuBar Property (InternetExplorer Object)

*object.MenuBar* [= *value*]

Returns or sets a value that determines whether Internet Explorer's menu bar is visible or hidden.

*object*

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

*value*

Optional. A Boolean expression that determines whether the menu bar is visible. If **True**, the menu bar is visible; if **False**, it is hidden.

### **Applies To**

InternetExplorer

## **Name Property (InternetExplorer Object)**

*object.Name*

Returns a string that evaluates to the name of the Internet Explorer application; that is, "Microsoft Internet Explorer."

*object*

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

### **Applies To**

InternetExplorer

## **Parent Property**

*object*.Parent

Returns the form on which the web browser control is located, or the automation object supported by Internet Explorer.

*object*

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

### **Applies To**

InternetExplorer, WebBrowser

## **Path Property (InternetExplorer Object)**

*object.Path*

Returns a string that evaluates to the full pathname of the Internet Explorer application.

*object*

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

### **Applies To**

InternetExplorer

## StatusBar Property (InternetExplorer Object)

*object*.StatusBar [= *value*]

Returns or sets a value that determines whether the status bar is visible.

*object*

Required. An object expression that evaluates to an InternetExplorer object.

*value*

Optional. A Boolean expression that determines whether the status bar is visible. If **True**, the status bar is visible; if **False**, it is not.

### Applies To

InternetExplorer



## **StatusText Property (InternetExplorer Object)**

*object*.StatusText [= *value*]

Returns or sets the text for the status bar.

*object*

Required. An object expression that evaluates to an InternetExplorer object.

*value*

Optional. A string that evaluates to the text for the status bar.

### **Applies To**

InternetExplorer

## ToolBar Property (InternetExplorer Object)

*object*.ToolBar [= *value*]

Returns or sets a value that determines whether the toolbar is visible.

*object*

Required. An object expression that evaluates to an InternetExplorer object.

*value*

Optional. A Boolean expression the determines whether the toolbar is visible. If **True**, the toolbar is visible; if **False**, it is hidden.

### Applies To

InternetExplorer

## Top Property (WebBrowser Object)

*object*.Top [= *value*]

Returns or sets the distance between the internal top edge of the web browser control and the top edge of its container.

*object*

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

*value*

Optional. A long integer expression specifying distance.

The Top property is measured in units depending on the coordinate system of its container. The values for this property changes as the object is moved by the user or by code.

### **Applies To**

InternetExplorer, WebBrowser

## **TopLevelContainer Property**

*object*.TopLevelContainer

Returns a Boolean value indicating whether the given object is a top-level container.

*object*

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

### **Applies To**

InternetExplorer, WebBrowser

## **Type Property (WebBrowser Object)**

*object.Type*

Returns a string expression that specifies the type name of the contained document object.

*object*

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

### **Applies To**

InternetExplorer, WebBrowser

## Visible Property (InternetExplorer Object)

*object.Visible* [= *value*]

Returns or sets a value indicating whether Internet Explorer is visible or hidden.

*object*

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

*value*

Optional. A Boolean expression specifying the visible state of Internet Explorer. If **True**, show the window; if **False**, hide it.

### Applies To

InternetExplorer

## **Width Property (WebBrowser Object)**

*object.Width* [= *width*]

Returns or sets the horizontal dimension, in pixels, of the frame window that contains the web browser control.

*object*

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

*width*

Optional. A long integer value specifying the horizontal dimension of the frame window, in pixels.

### **Applies To**

[InternetExplorer](#), [WebBrowser](#)

## ClientToWindow Method (InternetExplorer Object)

*object*.**ClientToWindow** *pcx* *pcy*

Converts the client coordinates of a point to window coordinates. Client coordinates are relative to the upper-left corner of the client area; window coordinates are relative to the upper-left corner of a window.

*object*

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

*pcx*

Required. A long integer value that specifies the x-coordinate of the point in client coordinates. When **ClientToWindow** returns, this variable contains the x-coordinate of the point in window coordinates.

*pcy*

Required. A long integer value that specifies the y-coordinate of the point in client coordinates. When **ClientToWindow** returns, this variable contains the y-coordinate of the point in window coordinates.

### Applies To

InternetExplorer



## **GetProperty Method (InternetExplorer Object)**

*object.GetProperty szProperty, vtValue*

Retrieves the current value of a property associated with the given object.

*object*

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

*szProperty*

Required. A string expression that contains the name of the property to retrieve.

*vtValue*

Required. A variable that receives the current value of the property.

### **Applies To**

InternetExplorer

## **GoBack Method (WebBrowser Object)**

*object.GoBack*

Navigates to the previous item in the history list.

*object*

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

### **Applies To**

InternetExplorer, WebBrowser

## **GoForward Method (WebBrowser Object)**

*object.GoForward*

Navigates to the next item in the history list.

*object*

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

### **Applies To**

InternetExplorer, WebBrowser

## **GoHome Method (WebBrowser Object)**

*object.GoHome*

Navigates to the current home or start page.

*object*

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

### **Applies To**

InternetExplorer, WebBrowser

## **GoSearch Method (WebBrowser Object)**

*object.GoSearch*

Navigates to the current search page.

*object*

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

### **Applies To**

InternetExplorer, WebBrowser

## Navigate Method (WebBrowser Object)

*object.Navigate* URL [*Flags*,] [*TargetFrameName*,] [*PostData*,] [*Headers*,] [*Referrer*]

Navigates to the resource identified by a Universal Resource Locator (URL), or to the file identified by a full pathname.

### *object*

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

### *URL*

Required. A string expression that evaluates to the URL of the resource to display, or the full pathname of the file to display.

### *Flags*

Optional. A constant or value that specifies whether to add the resource to the history list, whether to read or write from the cache, and whether to display the resource in a new window. It can be a combination of these values:

| Constant                  | Value | Meaning   |
|---------------------------|-------|---|
| <b>navOpenInNewWindow</b> | 1     | Open the resource or file in a new window.          |
| <b>navNoHistory</b>       | 2     | Exclude the resource or file from the history list. |
| <b>navNoReadFromCache</b> | 4     | Do not read from the cache.                         |
| <b>navNoWriteToCache</b>  | 8     | Do not write from the cache.                        |

### *TargetFrameName*

Optional. A string expression that evaluates to the name of the frame in which to display the resource.

### *PostData*

Optional. Data to send with the HTTP POST transaction.

### *Headers*

Optional. A value that specifies the HTTP headers to send.

### *Referrer*

Optional. A string expression that evaluates to the URL of the referring document. The referring document is the document that contains the link to the given document.

### **Applies To**

InternetExplorer, WebBrowser

## **PutProperty Method (InternetExplorer Object)**

*object.PutProperty szProperty, vtValue*

Sets the value of a property associated with the given object.

*object*

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

*szProperty*

Required. A string expression that contains the name of the property to set.

*vtValue*

Required. A variable that specifies the new value of the property.

### **Applies To**

InternetExplorer

## **Quit Method (InternetExplorer Object)**

*object.Quit*

Closes the Internet Explorer application.

*object*

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

### **Applies To**

InternetExplorer



## **Refresh Method (WebBrowser Object)**

*object.Refresh*

Reloads the file that the web browser control is currently displaying.

*object*

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

### **Applies To**

WebBrowser, InternetExplorer

## Refresh2 Method (WebBrowser Object)

*object.Refresh2 [Level]*

Reloads the file that the web browser control is currently displaying.

*object*

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

*Level*

Optional. Can be one of these constants or values:

| Constant                      | Value | Meaning  |
|-------------------------------|-------|--|
| <b>refreshAll</b>             | 0     | Refresh entirely.  |
| <b>refreshDontSendNoCache</b> | 1     | Do not send the HTTP header called pragma:nocache. This header tells the server not to return a cached copy, but to make sure the information is as fresh as possible. Browsers typically send this header when the user selects refresh, but the header causes problems for some servers. |

### Applies To

WebBrowser, InternetExplorer

## **Stop Method (WebBrowser Object)**

*object*.Stop

Cancels any pending navigation or download operation.

*object*

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

### **Applies To**

WebBrowser, InternetExplorer

## OnBeginNavigate Event (WebBrowser Object)

**Private Sub** *object*\_**OnBeginNavigate**(**ByVal** *URL* **As** **String**, **ByVal** *Flags* **As** **Long**, **ByVal** *TargetFrameName* **As** **String**, *PostData* **As** **Variant**, **ByVal** *Headers* **As** **String**, **ByVal** *Referrer* **As** **String**, *Cancel* **As** **Boolean**)

Occurs when the web browser control is about to navigate to a new hyperlink.

### *object*

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

### *URL*

A string expression that evaluates to the URL of the hyperlink to which the browser is navigating.

### *Flags*

A constant or value that specifies whether to add the resource to the history list, whether to read or write from the cache, and whether to display the resource in a new window. It can be a combination of these values:

| Constant                  | Value | Meaning   |
|---------------------------|-------|---|
| <b>navOpenInNewWindow</b> | 1     | Open the resource or file in a new window.          |
| <b>navNoHistory</b>       | 2     | Exclude the resource or file from the history list. |
| <b>navNoReadFromCache</b> | 4     | Do not read from the cache.                         |
| <b>navNoWriteToCache</b>  | 8     | Do not write from the cache.                        |

### *TargetFrameName*

Optional. A string expression that evaluates to the name of the frame in which to display the resource.

### *PostData*

Optional. Data to send with the HTTP POST transaction.

### *Headers*

Optional. A value that specifies the HTTP headers to be sent.

### *Referrer*

A string expression that evaluates to the URL of the referring document. The referring document is the document that contains the link to the given document.

### *Cancel*

A Boolean value that is True if the navigation operation was canceled, or False if it was not.

### **Applies To**

[InternetExplorer](#), [WebBrowser](#)

## **OnCommandStateChange Event (WebBrowser Object)**

**Private Sub *object*\_OnCommandStateChange (ByVal *Command* As Long, ByVal *Enable* As Boolean)**

Occurs when the enabled state of a command changes.

### *Object*

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

### *Command*

A long integer specifying the identifier of the command that changed.

### *Enable*

A Boolean value that is **True** if the command is enabled, or **False** if not.

### **Applies To**

InternetExplorer, WebBrowser

## **OnDownloadBegin Event (WebBrowser Object)**

**Private Sub** *object***\_OnDownloadBegin ( )**

Occurs when a new page is about to be downloaded.

### *Object*

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

### **Applies To**

InternetExplorer, WebBrowser

## **OnDownloadComplete Event (WebBrowser Object)**

**Private Sub** *object*\_**OnDownloadComplete** ( )

Occurs when the current page has finished being downloaded.

### *Object*

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

### **Applies To**

InternetExplorer, WebBrowser

## OnNavigate Event (WebBrowser Object)

**Private Sub WebBrowser1\_OnNavigate(ByVal URL As String, ByVal Flags As Long, ByVal TargetFrameName As String, PostData As Variant, ByVal Headers As String, ByVal Referrer As String)**

Occurs when the browser navigates to a new hyperlink.

### *object*

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

### *URL*

A string expression that evaluates to the URL of the hyperlink.

### *Flags*

A constant or value that specifies whether to add the resource to the history list, whether to read or write from the cache, and whether to display the resource in a new window. It can be a combination of these values:

| Constant                  | Value | Meaning   |
|---------------------------|-------|---|
| <b>navOpenInNewWindow</b> | 1     | Open the resource or file in a new window.          |
| <b>navNoHistory</b>       | 2     | Exclude the resource or file from the history list. |
| <b>navNoReadFromCache</b> | 4     | Do not read from the cache.                         |
| <b>navNoWriteToCache</b>  | 8     | Do not write from the cache.                        |

### *TargetFrameName*

Optional. A string expression that evaluates to the name of the frame in which to display the resource.

### *PostData*

Optional. Data to send with the HTTP POST transaction.

### *Headers*

Optional. A value that specifies the HTTP headers to send.

### *Referrer*

A string expression that evaluates to the URL of the referring document. The referring document is the document that contains the link to the given document.

### **Applies To**

InternetExplorer, WebBrowser



## OnNewWindow Event (WebBrowser Object)

**Private** *object\_OnNewWindow* (ByVal *URL* As String, ByVal *Flags* As Long, ByVal *TargetFrameName* As String, *PostData* As Variant, ByVal *Headers* As String, ByVal *Referrer* As String)

Occurs when the web browser control is about to create a new window for displaying information.

### *object*

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

### *URL*

A string expression that evaluates to the URL of the resource to be displayed in the new window.

### *Flags*

A constant or value that specifies whether to add the resource to the history list, whether to read or write from the cache, and whether to display the resource in a new window. It can be a combination of these values:

| Constant                  | Value | Meaning   |
|---------------------------|-------|---|
| <b>navOpenInNewWindow</b> | 1     | Open the resource or file in a new window.          |
| <b>navNoHistory</b>       | 2     | Exclude the resource or file from the history list. |
| <b>navNoReadFromCache</b> | 4     | Do not read from the cache.                         |
| <b>navNoWriteToCache</b>  | 8     | Do not write from the cache.                        |

### *TargetFrameName*

Optional. A string expression that evaluates to the name of the frame in which to display the resource.

### *PostData*

Optional. Data to send with the HTTP POST transaction.

### *Headers*

Optional. A value that specifies the HTTP headers to send.

### *Referrer*

Optional. A string expression that evaluates to the URL of the referring document. The referring document is the document that contains the link to the given document.

### **Applies To**

InternetExplorer, WebBrowser

## **OnProgress Event (WebBrowser Object)**

**Private Sub *object*\_OnProgress(ByVal *Progress* As Long, ByVal *ProgressMax* As Long)**

Occurs when the progress of a download operation is updated.

*object*

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

*Progress*

A long integer that specifies the number of bytes downloaded so far during the download operation.

*ProgressMax*

A long integer that specifies the total number of bytes that will be downloaded, if known; otherwise, this is zero.

The container can use the information provided by this event to display the number of bytes downloaded so far or to update a progress indicator.

### **Applies To**

InternetExplorer, WebBrowser

## **OnPropertyChange Event (InternetExplorer Object)**

**Private Sub *object*\_OnPropertyChange(ByVal *szProperty* As String)**

Occurs when the PutProperty method changes the value of a property.

*object*

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

*szProperty*

A string expression that contains the name of the property whose value has changed.

### **Applies To**

InternetExplorer

## **OnQuit Event (InternetExplorer Object)**

**Private Sub** *object*\_**OnQuit**(*Cancel* As Boolean)

Occurs when the Internet Explorer application is ready to quit.

*object*

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

*Cancel*

A Boolean value that is **True** if the last Quit was canceled, or **False** if not.

### **Applies To**

InternetExplorer

## **OnStatusTextChange Event (WebBrowser Object)**

**Private Sub *object*\_OnStatusTextChange(ByVal *bstrText* As String)**

Occurs when the status bar text has changed.

*object*

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

*bstrText*

A string containing the new status bar text.

The container can use the information provided by this event to update the text of a status bar.

### **Applies To**

InternetExplorer, WebBrowser

## **OnWindowActivated Event (InternetExplorer Object)**

**Private Sub** *object***\_OnWindowActivated( )**

Occurs when Internet Explorer's main window has been activated.

*object*

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

### **Applies To**

InternetExplorer

## **OnWindowMove Event (InternetExplorer Object)**

**Private Sub** *object***\_OnWindowMove( )**

Occurs when Internet Explorer's main window has been moved.

*object*

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

### **Applies To**

InternetExplorer

## **OnWindowSized Event (InternetExplorer Object)**

**Private Sub** *object***\_OnWindowSized( )**

Occurs when the size of Internet Explorer's main window has changed.

*object*

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

### **Applies To**

InternetExplorer



## Microsoft Internet Explorer

The Microsoft® Internet Explorer scripting object model is used by scripts that enliven World Wide Web content. The Internet Explorer object model is compatible with the object model used in the JavaScript (TM) language. However, this object model is accessible not only from JavaScript but also from any scripting language that plugs into the ActiveX scripting framework, such as the Microsoft Visual Basic® Scripting Edition (VBScript) language. This document provides an overview of the object model, sample code (in both JavaScript and VBScript), and reference information.

This document outlines the methods, properties, and events available to scripting engines in Internet Explorer. This model has been designed to provide maximum compatibility with Netscape for existing pages, while easing the transition to Visual Basic Script for Visual Basic developers.

**Note** All properties and methods that modify the HTML contents must be called during HTML parse time. This means that the code must reside in a script block that will run inline during the loading of the HTML document. This is called *immediate execution* in the ActiveX Scripting Model.

## **Attaching and Invoking Scripts**

There are three ways to attach and invoke scripts in HTML: contain them in the `<SCRIPT>` tag, use attributes of HTML elements, or use a custom URL type.

## Using the SCRIPT Element

Use the SCRIPT element to add scripts to HTML. SCRIPT is a character-like element for embedding script code anywhere in the document HEAD or BODY. The SCRIPT element can be used to reference external scripts, using the SRC attribute, and to include script statements within the HTML document.

HTML documents can include multiple SCRIPT elements that can be placed in the document HEAD or BODY. This allows script statements for a form to be placed near the corresponding FORM element.

Here is a simple example of a page that uses the SCRIPT element:

```
<SCRIPT language="VBScript">  
    '... Additional VBScript statements ...  
</SCRIPT>
```

The same example in JavaScript would read:

```
<SCRIPT language="JavaScript">  
    //... Additional JavaScript statements ...  
</SCRIPT>
```

## Evaluation of SCRIPT

{ewc HLP95EN.DLL, DYNALINK, "See Also":"scriptom\_000000002010100C"}

The SCRIPT element is evaluated when the document is loaded, and all code is executed at load time. This has some side effects.

First, for functions like document.write, the order of script elements can affect the output of the document. For example, the page:

```
<HTML><BODY>
<SCRIPT LANGUAGE="JavaScript">
document.write ("Hello world.")
</SCRIPT>
This is a document.
</BODY></HTML>
```

results in:

Hello world. This is a document.

While the page:

```
<HTML><BODY>
This is a document.
<SCRIPT LANGUAGE="VBScript">
document.write ("Hello world.")
</SCRIPT>
</BODY></HTML>
```

results in:

This is a document. Hello world.

Second, because script statements are evaluated when the document is loaded, attempts to reference objects will fail if these objects are defined by HTML elements that occur later in the document.

Note that the document object's write method can insert not just text but also objects such as buttons (defined using the INPUT tag) and ActiveX controls (defined using the OBJECT tag.) Currently, these objects can not be referenced only in a script block following the script block that output them. Future betas of Internet Explorer will provide full Netscape comptability by allowing objects to be referenced as soon as they are output.

## Using Scripts as Attributes of HTML Elements

Another way to insert scripts is to add attributes to element tags in HTML. These attributes match with events on the elements, and the "scriptlet" is executed when the event is fired. This method can be used for any HTML intrinsic elements, such as forms, buttons, or links; however, this method does not work for items inserted using the OBJECT tag.

The following example uses this syntax in Button1 to handle the onClick event. To demonstrate the ability to combine multiple scripting languages on the same page, the scriptlet for Button1 is implemented in VBScript and that for Button2 in JavaScript.

```
<form name="Form1">
  <input type="button" name="Button1" value="  Press me  "
    onClick="pressed" language="VBScript">
  <input type="button" name="Button2" value="Press me too!"
    onClick="pressed2()" language="JavaScript">
</form>

<script language="VBSCRIPT">
  sub pressed
    document.Form1.Button1.value="I'm VBS"
    alert "I've been pressed!"
  end sub
</script>
<script language="JavaScript">
  function pressed2()
  {
    document.Form1.Button2.value="I'm JavaScript"
    alert("Are you impressed?")
  }
</script>
```

Notice the use of the language attribute on the input tag to indicate which language the scriptlet is in. If no language is specified, the scriptlet defaults to the language of the most recently encountered script block, or JavaScript if no script block has been encountered.

The elements FORM, INPUT, BODY, and A support this syntax, but with differing events. For detailed information, see the individual tags referenced later in this document.

## An Alternative Using SCRIPT

The SCRIPT element can also be used with the FOR="object" EVENT="eventname" syntax. This method can be used for any named elements, and for any elements inserted using the OBJECT tag. The following example is similar to the previous "scriptlet" example, but it uses a different syntax:

```
<form name="Form1">
  <input type="button" name="Button1" value="Press me">
  <script for="Button1" event="onClick" language="VBScript">
    alert "I've been pressed"
    document.Form1.Button1.value="OUCH"
  </script>
</form>
```

## Using Scripts in URLs

Scripts can be invoked using the A element combined with a custom URL type. This allows a script to be executed when the user clicks on a hyperlink. This URL type is valid in any context, but is most useful when used with the A element. For example:

```
<A HREF="javascript:alert('hi there')">Click me to see a message.</A>
```

displays an alert message box that contains the text 'hi there'.

### Syntax

*script-engine:script-code*

Executes the script code using the script engine when the URL is resolved. For example, to execute a script when the user clicks on a hyperlink, use:

```
<title> JavaScript example </title>
<A HREF=" javascript:alert(document.title)">Click here to see the title of
the current document..</A>
```

Notice that the script is executed in the context of the current page, which means that document.title evaluates to the document containing the script.

| Argument      | Type   | Description   |
|---------------|--------|---|
| script-engine | String | A string that names a scripting engine ( <i>must</i> be JavaScript for Beta 1).   |
| script-code   | String | A string that evaluates to a script in the syntax supported by the scripting engine. This script is executed by the scripting engine when the URL is evaluated. |

**Note** This syntax is only supported for JavaScript in the current build of Internet Explorer; in particular, vbscript: will not work in the current build. All scripting engines will be supported in future builds. Also, the JavaScript: syntax is currently supported only from scripts, not when typed into the address bar by users.

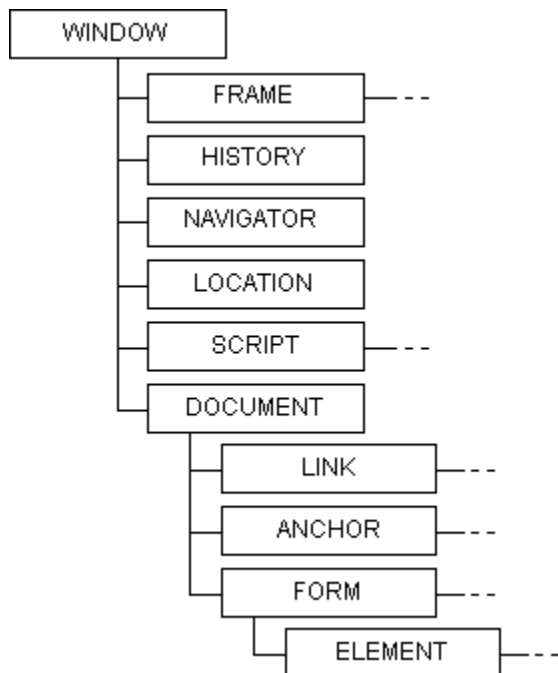
## Object Hierarchy and Scoping

{ewc HLP95EN.DLL, DYNALINK, "See Also": "scriptom\_000000003000000C"}

There are eleven objects to consider in the HTML object model:

- Window
- **Frame**
- History
- Navigator
- Location
- **Script**
- Document
- Link
- Anchor
- Form
- Element

These objects are organized in the following hierarchy (the dotted line following an object indicates that multiple objects may exist):



Each of these objects has its own rules for scoping and containment.



## The Window Object

```
{ewc HLP95EN.DLL, DYNALINK, "See Also":"scriptom_0000000003010000C"}
```

The top level object is a window. Every window contains:

- **Frame** - Array of contained frame windows. Each frame is a window that has its own properties, including a document.
- History - History object for the current window. This object is used to access the history list from the browser.
- Navigator - Navigator object for the current window. The navigator object contains information about the browser application.
- Location - Location object for the current window. Provides information about the location of the window's URL.
- **Script** - Any scripting function defined using the SCRIPT element in the window scope.
- Document - document in the current window.

The window object properties can be referenced directly by scripts while in the window scope. So, for example, script authors do not need to type:

```
window.name
```

to reference the window name; instead, it is sufficient just to type:

```
name
```

Note also that it is possible to call scripts from one window object to another. So, to execute the script `myscript` in the topmost window, use:

```
top.myscript()
```

## The Document Object

```
{ewc HLP95EN.DLL, DYNALINK, "See Also": "scriptom_000000003020000C"}
```

The next level down is a document. This object contains:

- Link - an array of hyperlinks found on the given document.
- Anchor - an array of forms found on the given document
- Form - an array of anchors found on the given document

Because scripts live with the window object, not the document object, the script author must type **document.property** to access document properties. So, to get the title of the document, the author can type:

```
<script language="VBScript">
'...
string1 = document.title      -put the document title into string1
'...
</script>
```

To access the forms in a document, the author can either refer by name or through the form array. So, for the following form:

```
<form name="Form1">
  <input type="button" name="Button1" value="Press ME" onClick="pressed">
</form>
```

The author can access the object named button1 either by name:

```
<script language="VBScript">
sub pressed
  document.Form1.Button1.value="I've been pressed" ' access the form by
name
end sub
</script>
```

or by index:

```
<script language="VBScript">
sub pressed
  document.forms(0).Button1.value="I've been pressed" ' access the
form by index
end sub
</script>
```

The only unusual part of document naming is contained elements that are not form types. Scripts can refer to these elements directly, without using **document**. So, for example, if the authors create an object called myObject, they can reference it directly in script as follows:

```
<object name="myObject" ... >
</object>

<script language="VBScript">
sub foo
  myObject.color = "green" - access the form by index
end sub
</script>
```

## The Form Object

```
{ewc HLP95EN.DLL, DYNALINK, "See Also": "scriptom_000000003030000C"}
```

The final level of scoping is a form. The form object contains:

- Element - the array of objects and intrinsic controls contained in the form.

Scripts can live either in a form object or in a window object. If a script lives outside the form, it needs to access the elements in the form, either by name or through the form array (see the example in "The Document Object"). If, however, the script lives inside the form, it can access the elements in the form directly.

```
<form name="Form1">
  <input type="button" name="Button1" value="Press me">
  <script for="Button1" event="onClick" language="VBScript">
    alert "I've been pressed"
    document.Form1.Button1.value="OUCH"      - as usual, we can use the
fully qualified name
    Button1.value="OUCH"      - because we're in the form Button1 is
scoped as well
  </script>
</form>

<script language="VBScript">
sub foo
  document.Form1.Button1.value="OUCH"      - outside the form, we can only
use the fully qualified name
end sub
</script>
```

## window Object

The top level object in the scripting object model is a window. Every window contains:

- **Frame** - Array of frame windows contained by a parent window. Each frame is a window that has its own properties, including a document.
- History - the history object for the current window. This object is used to access the history list from the browser.
- Navigator - the navigator object for the current window. The navigator object contains information about the browser application.
- location - the location object for the current window. Provides information about the location of the window's URL.
- **Script** - any scripting function defined using the SCRIPT element in the window scope.
- Document - the document in the current window.



The window object represents the Internet Explorer window and its methods and properties. Methods and properties of the window object can be called by scripts directly. This means that if you wanted to get the name of the current page, you would use the following script (Notice that the property name does not need a prefix):

```
<script language="VBScript">
    '...
    string1=name           - get the name of the current window
    alert string1         - display that name as an alert
    '...
</script>
```

However, you can access the properties of other window objects without explicitly mentioning the window. For example, to get the name of the current window's parent, you would use:

```
<script language="VBScript">
    '...
    string1=parent.name   - get the name of the parent window
    '...
</script>
```

window events can be hooked to scripts using extensions to the BODY tag. To add scripts to a window event, in the BODY tag at the top of the page, add a "scriptlet" for either the onLoad or onUnload events. In the following example, the *Foo* function is called when the page is loaded:

```
<HTML>
...
<BODY Language="VBS" onLoad="Foo">
...
<SCRIPT language="VBScript">
...
Sub Foo
    MsgBox "This is sub foo"
End Sub
...
</SCRIPT>
...
</BODY></HTML>
```

To access a window by name, the window must be given a name. This can happen in three ways: by using the *window.open* method, by creating the window with a name using the FRAMESET element, or by creating the window with a URL using the TARGET attribute.

The following examples all create a window named *foo* with contents *a.htm*.

```
<SCRIPT Language="VBScript">
window.open ( "a.htm", "foo");
</SCRIPT>
```

```
<FRAMESET cols = "200, *" frameborder=0>
  <FRAME name = "foo" src="a.htm">
  <FRAME name = "bar" src="b.htm">
</FRAMESET>
```

```
<A HREF="a.htm" TARGET = "foo">Click here to see a.htm in window foo.</A>
```

The current implementation of Internet Explorer does not support *window.open*.

### **Methods**

[alert](#), [confirm](#), [prompt](#), [open](#), [close](#), [setTimeout](#), [clearTimeout](#), [navigate](#)

### **Events**

[onLoad](#), [onUnload](#)

### **Properties**

[name](#), [parent](#), [self](#), [top](#), [location](#), [defaultStatus](#), [status](#), [frames](#), [history](#), [navigator](#), [document](#)

## Properties

{ewc HLP95EN.DLL, DYNALINK, "See Also":"scriptom\_000000004010000C"}

window properties can be referenced directly in the scripting language. Consequently, all window properties are reserved words and cannot be used as variable names in procedures. The following window properties are used:

name, parent, self, top, location, defaultStatus, status, frames, history, navigator, document

## **name Property**

*[window.]name*

Returns the name of the current window.

- Returns the string containing the current window name. Note that the current implementation always returns "Microsoft Internet Explorer."

*window*

Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

To set the value of String1 to be the name of the current window, use:

```
String1=name.
```

This property is read-only.

### **Applies To**

Window

### **Methods**

alert, confirm, prompt, open, close, setTimeout, clearTimeout, navigate

### **Events**

onLoad, onUnload

### **Properties**

parent, self, top, location, defaultStatus, status, frames, history, navigator, document

## parent Property

[*window*.]parent

Returns the window object of the window's parent. This property is read-only. The parent of the window is the containing frame. If the current window has no containing frame windows, then the parent evaluates to the current window.

- Returns the window object that evaluates to the parent window.

*window*

Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

To set the value of String1 to be the name of the parent of the current window, use:

```
String1=parent.Name.
```

### Applies To

Window

### Methods

alert, confirm, prompt, open, close, setTimeout, clearTimeout, navigate

### Events

onLoad, onUnload

### Properties

name, self, top, location, defaultStatus, status, frames, history, navigator, document



## **self Property**

[*window*.]self

Returns the window object of the current window. This property is read-only.

- Returns an object that evaluates to the current window.

*window*

Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

To set the value of String1 to be the name of the current window, use:

```
String1=self.name
```

### **Applies To**

Window

### **Methods**

alert, confirm, prompt, open, close, setTimeout, clearTimeout, navigate

### **Events**

onLoad, onUnload

### **Properties**

name, parent, top, location, defaultStatus, status, frames, history, navigator, document

## top Property

[*window*.]top

Returns the window object of the topmost window. This property is read-only. The topmost window is the containing window of all frames in the current browser instance.

- Returns an object that evaluates to the topmost window.

*window*

Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

To set the value of String1 to be the name of the topmost window, use:

```
String1=top.name.
```

### Applies To

Window

### Methods

alert, confirm, prompt, open, close, setTimeout, clearTimeout, navigate

### Events

onLoad, onUnload

### Properties

name, parent, self, location, defaultStatus, status, frames, history, navigator, document

## location Property

[*window*.]location

Returns the location object for the current window. This property is read-only. For more details, see "Location Object."

- Returns an object that evaluates to the location object of window.

*window*

Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

To set the value of String1 to be the name of the URL of the current window, use:

```
String1=location.HRef.
```

### Applies To

Window

### Methods

alert, confirm, prompt, open, close, setTimeout, clearTimeout, navigate

### Events

onLoad, onUnload

### Properties

name, parent, self, top, defaultStatus, status, frames, history, navigator, document

## defaultStatus Property

[*window*.]defaultStatus[=*string*]

Gets or sets the default status text in the lower left portion of the status bar.

- Returns the default status text.

*window*

Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

*String*

Optional. Sets the default status text to the value of String.

To set the default status to "Hello," use:

```
defaultStatus="Hello"
```

Note that this property does not currently set the default status message, so it is the same as calling **status**.

### Applies To

Window

### Methods

alert, confirm, prompt, open, close, setTimeout, clearTimeout, navigate

### Events

onLoad, onUnload

### Properties

name, parent, self, top, location, status, frames, history, navigator, document

## status Property

[*window*.]status[=*string*]

Gets or sets the status text in the lower left of the status bar.

- Returns the current status text.

*window*

Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

*String*

Optional. Sets the status text to the value of String.

To set the status to "Hello," use:

```
status="Hello."
```

Currently not implemented.

### Applies To

Window

### Methods

alert, confirm, prompt, open, close, setTimeout, clearTimeout, navigate

### Events

onLoad, onUnload

### Properties

name, parent, self, top, location, defaultStatus, frames, history, navigator, document

## frames Property

[*window*].frames[*integer*]

Returns the array of frames for the current window.

- Returns an object expression that evaluates to the array of frames.

*window*

Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

To set String1 to the URL of frame[0], use:

```
String1=parent.frames[0].location.href.
```

### Applies To

Window

### Methods

alert, confirm, prompt, open, close, setTimeout, clearTimeout, navigate

### Events

onLoad, onUnload

### Properties

name, parent, self, top, location, defaultStatus, status, history, navigator, document

## history Property

[*window*.]history

Returns the history object of the current window. For more details on methods, properties, and events, see "history Object."

- Returns an object expression that evaluates to a history object.

*window*

Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

### Applies To

Window

### Methods

alert, confirm, prompt, open, close, setTimeout, clearTimeout, navigate

### Events

onLoad, onUnload

### Properties

name, parent, self, top, location, defaultStatus, status, frames, navigator, document

## **navigator Property**

`[window.]navigator`

Returns the navigator object of the current window. For more details on methods, properties, and events, see "navigator Object."

- Returns an object expression that evaluates to a navigator object.

*window*

Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

### **Applies To**

Window

### **Methods**

alert, confirm, prompt, open, close, setTimeout, clearTimeout, navigate

### **Events**

onLoad, onUnload

### **Properties**

name, parent, self, top, location, defaultStatus, status, frames, history, document



## document Property

[*window*.]document

Returns the document object of the current window. For more details on methods, properties, and events, see "document Object."

- Returns an object expression that evaluates to a document object.

*window*

Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

### Applies To

Window

### Methods

alert, confirm, prompt, open, close, setTimeout, clearTimeout, navigate

### Events

onLoad, onUnload

### Properties

name, parent, self, top, location, defaultStatus, status, frames, history, navigator

## **Methods**

This section describes the methods for the window object.

## **alert Method**

*[window.]alert string*

Displays an alert message box.

*window*

Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

*String*

A string containing the text to display in the message box.

The following example would display an alert that contained the string "Hello World":

```
Alert "Hello World"
```

### **Applies To**

Window

### **Methods**

confirm, prompt, open, close, setTimeout, clearTimeout, navigate

### **Events**

onLoad, onUnload

### **Properties**

name, parent, self, top, location, defaultStatus, status, frames, history, navigator, document

## **confirm Method**

*[bool =][window.]confirm string*

Displays a message box that allows the user to select **OK** or **Cancel** and returns either TRUE or FALSE.

- Returns the user response: TRUE if the user pressed OK; FALSE if not.

*window*

Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

*String*

A string containing the text to display in the message box.

The following example would display a message box that contained the string "Do you want to continue?":

```
x=Confirm "Do you want to continue?"
```

### **Applies To**

Window

### **Methods**

alert, prompt, open, close, setTimeout, clearTimeout, navigate

### **Events**

onLoad, onUnload

### **Properties**

name, parent, self, top, location, defaultStatus, status, frames, history, navigator, document

## **prompt Method**

*[string* **=**]*[window.]prompt* [*prompt*] [, *default*]

Prompts the user for input.

- Returns the user input.  
Not implemented in current build.

*window*

Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

*String*

Optional. A string containing the text to display in the message box.

*default*

Optional. A string containing the default text to display in the input field.

### **Applies To**

Window

### **Methods**

alert, confirm, open, close, setTimeout, clearTimeout, navigate

### **Events**

onLoad, onUnload

### **Properties**

name, parent, self, top, location, defaultStatus, status, frames, history, navigator, document

## open Method

`[newwindow = ][window.]open url, target, ["[toolbar=bool] [, location=bool][, directories=bool][, status=bool][, menubar=bool][, scrollbars=bool][, resizable=bool][, width=pixels][, height=pixels]"`

Creates a new window.

- Returns an object expression that evaluates to the created window object.

### *window*

Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

### *url*

A string containing a correctly parsed URL. The URL is parsed identically to a link—both relative and absolute paths are supported.

### *target*

A string containing the name of the target window. If a window with this name already exists, the existing window is reused with the new URL. If the window does not exist, a new window is created with that name. Note that this works identically to the TARGET attribute of an HREF in HTML.

### *bool*

The remaining window properties are passed as a comma-separated list. Most of these can be set to Boolean values, either *yes* or *no* [1 or 0]. These properties are toolbar, location, directories, status, menubar, scrollbars, and resizable.

### *pixels*

Two other properties in this list, width and height, have values in pixels.

The following example would create a new window:

```
open "http://www.microsoft.com", "myWindow", "toolbar=no, location=no, directories=no"
```

Note: This feature is not currently implemented in Internet Explorer.

## Applies To

Window

## Methods

alert, confirm, prompt, close, setTimeout, clearTimeout, navigate

## Events

onLoad, onUnload

## Properties

name, parent, self, top, location, defaultStatus, status, frames, history, navigator, document

## **close Method**

`[window.]close`

Closes the window.

- Returns an object expression that evaluates to the indexed frame.

*window*

Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

Note: This feature is not currently implemented in Internet Explorer.

### **Applies To**

Window

### **Methods**

alert, confirm, prompt, open, setTimeout, clearTimeout, navigate

### **Events**

onLoad, onUnload

### **Properties**

name, parent, self, top, location, defaultStatus, status, frames, history, navigator, document

## setTimeout Method

**ID** = [*window*.]setTimeout *expression*, *msec*

Sets a timer to call a function after a specified number of milliseconds.

- Returns the ID of the timer object. This can be used to cancel the timer using the **clearTimeout** method.

### *window*

Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

### *Expression*

An object expression that evaluates to a function or object property. This function is called when the Timeout is set.

### *MSec*

The number of milliseconds that passes before the expression is evaluated.

To call Button1.Click after 100 milliseconds, use:

```
MyID = setTimeout ("Button1.Click", 100).
```

## Applies To

Window

## Methods

alert, confirm, prompt, open, close, clearTimeout, navigate

## Events

onLoad, onUnload

## Properties

parent, self, top, location, defaultStatus, status, frames, name, history, navigator, document



## **clearTimeout Method**

`[window.]clearTimeout ID`

Clears the timer having a particular ID.

*window*

Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

*ID*

The ID of the timer to be cleared. If there is no timer with this ID, the function does nothing. To clear the timer with ID=MyID, use.

```
clearTimeout MyID
```

### **Applies To**

Window

### **Methods**

alert, confirm, prompt, open, close, setTimeout, navigate

### **Events**

onLoad, onUnload

### **Properties**

parent, self, top, location, defaultStatus, status, frames, name, history, navigator, document

## **navigate Method**

*[window.]navigate url*

Navigates the window to a new URL.

*window*

Optional. An object expression that evaluates to a window object. If omitted, the current script window is used.

*url*

A string containing a valid URL. This can be either relative or absolute.

### **Applies To**

Window

### **Methods**

alert, confirm, prompt, open, close, setTimeout, clearTimeout

### **Events**

onLoad, onUnload

### **Properties**

name, parent, self, top, location, defaultStatus, status, frames, history, navigator, document

## **onLoad Event**

**onLoad**=*function-name*

Fired when the contents of the window are loaded.

*function-name*

An object expression that evaluates to a scripting function.

To call the VBS function Foo when the page is loaded, use:

```
<BODY Language="VBS" onLoad="Foo">
```

### **Applies To**

Window

## **onUnload Event**

**onUnload**=*function-name*

Fired when the contents of the window are unloaded.

*function-name*

An object expression which evaluates to a scripting function.

To call the VBS function Foo when the page is unloaded, use:

```
<BODY Language="VBS" onUnload="Foo">[<window.>]Navigate url
```

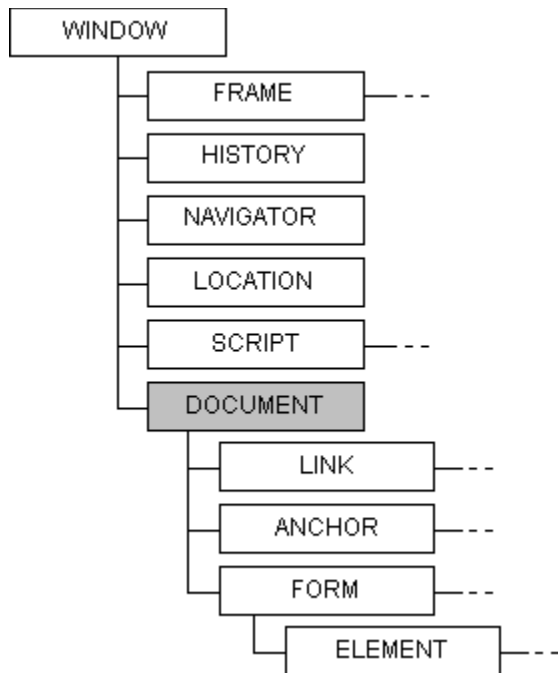
### **Applies To**

Window

## document Object

An object that resides below the window in the scripting object model. A document may contain:

- Link - an array of hyperlinks found on the given document
- Anchor - an array of anchors found on the given document
- Form - an array of forms found on the given document



The document object reflects the HTML document currently in the browser and objects on the page—that is, links, forms, buttons, and ActiveX Objects. Methods and properties of the document object must be called in a script by placing *document* first in the statement. This means that if you wanted to set the background color on the page, the script would look like:

```
<script language="VBScript">
    document.bgColor = "Blue"
</script>
```

The document object currently has no events.

### Methods

write, writeln, open, close, clear

### Properties

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, anchors, links, forms, location, lastModified, title, cookie, referrer

## linkColor Property

`document.linkColor` [=rgb-value|string]

Gets or sets the current color of the links in a document.

- Returns the rgb value of the current link color.

*document*

An object expression that evaluates to a document object.

*rgb-value*

Optional. The new color of links in the document.

*string*

Optional. A string value specifying the color.

Note that this property can only be set at parse time, not after the page is painted. So the code:

```
<SCRIPT LANGUAGE="JavaScript">
document.vLinkColor = "green";
document.linkColor = "red";
document.alinkColor = "aqua";
</SCRIPT>
```

sets the link color, while the code:

```
<FORM>
"document.linkColor='#000000'">
<INPUT TYPE="button" VALUE="Set Visited Link Color to White" onClick =
"document.vLinkColor='#FFFFFF'">
</FORM>
```

will have no effect when the button is clicked. The performance hit of changing the link color after parse time is simply too great.

### Applies To

Document

### Methods

write, writeln, open, close, clear

### Properties

aLinkColor, vLinkColor, bgColor, fgColor, anchors, links, forms, location, lastModified, title, cookie, referrer

## **aLinkColor Property**

*document.aLinkColor* [=rgb-value|string]

Gets or sets the current color of the *active* links in a document. A link is *active* when the mouse pointer is held down over the link but not released. Note that Internet Explorer does not have this feature, so **aLinkColor** has no effect; however, it is supported in the object model for compatibility reasons. As with **linkColor**, this property can only be set at parse time. For details, see the examples in **linkColor**.

- Returns the rgb value of the current link color.

*document*

An object expression that evaluates to a document object.

*rgb-value*

Optional. The new color of links in the document.

*string*

Optional. A string value specifying the color.

### **Applies To**

Document

### **Methods**

write, writeLn, open, close, clear

### **Properties**

linkColor, vLinkColor, bgColor, fgColor, anchors, links, forms, location, lastModified, title, cookie, referrer

## **vLinkColor Property**

*document.vLinkColor* [=rgb-value|string]

Gets or sets the current color of the visited links in a document. As with **linkColor**, this property can only be set at parse time. See the examples in **linkColor** for details.

- Returns the rgb value of the current link color.

*document*

An object expression that evaluates to a document object.

*rgb-value*

Optional. The new color of links in the document.

*string*

Optional. A string value specifying the color.

### **Applies To**

Document

### **Methods**

write, writeLn, open, close, clear

### **Properties**

linkColor, aLinkColor, bgColor, fgColor, anchors, links, forms, location, lastModified, title, cookie, referrer



## bgColor Property

`document.bgColor` [=rgb-value|string]

Gets or sets the current color of the background in a document.

- Returns the rgb value of the current background color.

*document*

An object expression that evaluates to a document object.

*rgb-value*

Optional. The new color of the background in the document.

*string*

Optional. A string value specifying the color.

To set the background color to white, use:

```
document.bgColor="000000"
```

### Applies To

Document

### Methods

write, writeLn, open, close, clear

### Properties

linkColor, aLinkColor, vLinkColor, fgColor, anchors, links, forms, location, lastModified, title, cookie, referrer

## **fgColor Property**

*document*.fgColor[=*rgb-value*]

Gets or sets the foreground color.

*document*

An object expression that evaluates to a document object.

*rgb-value*

Optional. The new color of the foreground in the document.

To set the foreground color to white, use:

```
document.fgColor="000000".
```

### **Applies To**

Document

### **Methods**

write, writeLn, open, close, clear

### **Properties**

linkColor, aLinkColor, vLinkColor, bgColor, anchors, links, forms, location, lastModified, title, cookie, referrer

## anchors Property

*document.anchors[integer]*

Returns the array of anchors in a document.

- Returns an object expression that evaluates to the array of anchors.

*document*

An object expression that evaluates to a document object.

To access the first anchor in the document, use:

```
document.anchors[0]
```

To get the length of the anchors array, use:

```
document.anchors.length
```

### Applies To

Document

### Methods

write, writeLn, open, close, clear

### Properties

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, links, forms, location, lastModified, title, cookie, referrer

## links Property

`document.links` [*integer*]

Returns the array of links for the current document.

- Returns an object expression that evaluates to the array of links.

*document*

An object expression that evaluates to a document object.

To access the first link in the document, use:

```
document.Links[0]
```

To get the length of the links array, use:

```
document.links.length
```

Note that the locations in the links collection are read-only in the current build. In future builds you will be able to reset the targets of links.

### Applies To

Document

### Methods

write, writeLn, open, close, clear

### Properties

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, anchors, forms, location, lastModified, title, cookie, referrer

## forms Property

`document.forms` [*integer*]

Returns the array of forms in a document.

- Returns an object expression that evaluates to the array of forms.

*document*

An object expression that evaluates to a document object.

To access the first form in the document, use:

```
document.Forms[0]
```

To get the length of the forms array, use:

```
document.forms.length
```

### Applies To

Document

### Methods

write, writeLn, open, close, clear

### Properties

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, anchors, links, location, lastModified, title, cookie, referrer

## location Property

*document.location*

Returns a read-only representation of the location object.

- Returns an object expression that evaluates to the location object of the document.

*document*

An object expression that evaluates to a document object.

To set String1 to the document's URL, use:

```
String1 = document.location.Href
```

### Applies To

Document

### Methods

write, writeln, open, close, clear

### Properties

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, anchors, links, forms, lastModified, title, cookie, referrer

## **lastModified Property**

*document.lastModified*

Returns the last modified date of the current page.

- Returns a string containing the date.

*document*

An object expression that evaluates to a document object.

To set Date1 to the document's URL, use:

```
Date1 = document.lastModified
```

### **Applies To**

Document

### **Methods**

write, writeLn, open, close, clear

### **Properties**

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, anchors, links, forms, location, title, cookie, referrer

## **title Property**

*document.title*

Returns a read-only representation of the document's title.

- Returns an object expression that evaluates to the location object of the document.

*document*

An object expression that evaluates to a document object.

To set String1 to the document's title, use:

```
String1 = document.title
```

### **Applies To**

Document

### **Methods**

write, writeln, open, close, clear

### **Properties**

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, anchors, links, forms, location, lastModified, cookie, referrer



## cookie Property

`document.cookie` [=newcookie]

Gets or sets the cookie for the current document.

- Returns a string containing the current cookie.

*document*

An object expression that evaluates to a document object.

*newcookie*

Optional. The new value for the cookie. Because the cookie file is just a text file, this value is a string.

The cookie is a string expression stored for the current page. Note that setting the cookie overwrites any current cookie information. Also note that you can use string expressions to locate particular information in the cookie string.

### Applies To

Document

### Methods

write, writeLn, open, close, clear

### Properties

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, anchors, links, forms, location, lastModified, title, referrer

## referrer Property

*document.referrer*

Gets the URL of the referring document.

- Returns a string containing the URL of the referring document.  
Currently returns the URL of the referring document when there is a referrer, and NULL when there is no referrer.

*document*

An object expression that evaluates to a document object.

The referring document is the document that contained the link the user clicked on to get to the current document. For example, if the user is on [www.microsoft.com](http://www.microsoft.com) and clicks on a link to navigate to [www.msn.com](http://www.msn.com), the referrer property of the document for [www.msn.com](http://www.msn.com) is [www.microsoft.com](http://www.microsoft.com). Note that by definition the referrer varies depending on how the user linked to the current document. If the user navigated to the document without clicking on a link from another page, referrer should return NULL.

### Applies To

Document

### Methods

write, writeLn, open, close, clear

### Properties

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, anchors, links, forms, location, lastModified, title, cookie

## write Method

*document.writestring*

Places the given string into the current document. Unless otherwise specified, the string is appended to the current document at the current position.

*document*

An object expression that evaluates to a document object.

*string*

The string to write to the current document. Note that the string is added into the HTML directly, so it must be formatted as HTML.

The following examples demonstrate the use of the **write** method:

```
<HTML><BODY>
<SCRIPT LANGUAGE='VBS'>
document.Write ("Hello world.")
</SCRIPT>
This is a document.
</BODY></HTML>
```

results in:

Hello world. This is a document.

Whereas:

```
<HTML><BODY>
This is a document.
<SCRIPT LANGUAGE='VBS'>
document.Write ("Hello world.")
</SCRIPT>
</BODY></HTML>
```

results in:

This is a document. Hello world.

### Applies To

Document

### Methods

writeLn, open, close, clear

### Properties

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, anchors, links, forms, location, lastModified, title, cookie, referrer

## **writeln Method**

*document.writeln string*

Places the given string into the current document with a new-line character appended to the end.

*document*

An object expression that evaluates to a document object.

*string*

The string to write to the current document. Note that the string is added into the HTML directly, so it must be formatted as HTML.

This method is the same as the *document.write* method with the addition of a newline character at the end. Note that a newline is ignored by HTML unless it is bracketed by <PRE> tags, so in many cases *document.write* and *document.writeln* behave exactly the same.

The following examples demonstrate the use of the **writeln** method:

```
<SCRIPT LANGUAGE='VBS'>
document.writeln ("Hello world.")
document.write ("Hello world.")
</SCRIPT>
```

**results in:**

Hello world. Hello world.

**Whereas:**

```
<PRE>
<SCRIPT LANGUAGE='VBS'>
document.writeln ("Hello world.")
document.write ("Hello world.")
</SCRIPT>
</PRE>
```

**results in:**

Hello world.  
Hello world.

### **Applies To**

Document

### **Methods**

write, open, close, clear

### **Properties**

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, anchors, links, forms, location, lastModified, title, cookie, referrer

## open Method

*document.open* [*contentType*]

Opens the document stream for output.

*document*

An object expression that evaluates to a document object.

*contentType*

Optional. A string containing a valid mime type. Note that this can include types supported by Internet Explorer (*text/html*, *text/plain*), but can also include other types (*application/x-director* for MacroMedia director). In the case of other types, Internet Explorer creates a file containing the data between *document.open* and *document.close* calls, and hands the created file to the correct application.

This method is implemented in the current build; however, the mime-type is currently ignored (*html* is always assumed).

Generally *document.open* is followed by a sequence of *document.write* or *document.writeln* statements, followed by *document.close*. If the referenced document exists already, any information contained in the document is cleared. To write "Hello World" to the document, use:

```
document.open
document.writeln "Hello World"
document.close
```

Note that this is identical to:

```
document.writeln "Hello World"
```

with two exceptions.

- In the first example, "Hello World" is written to the screen after *document.close*; in the second, it is written immediately.
- In the first example, *document.open* clears the document if there is data; in the second, "Hello world" is appended to the end.

### Applies To

Document

### Methods

write, writeln, close, clear

### Properties

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, anchors, links, forms, location, lastModified, title, cookie, referrer

## **close Method**

*document.close*

Updates the screen to display all of the strings written after the last open method call.

*document*

An object expression that evaluates to a document object.

### **Applies To**

Document

### **Methods**

write, writeLn, open, clear

### **Properties**

linkColor, aLinkColor, vLinkColor, bgColor, fgColor, anchors, links, forms, location, lastModified, title, cookie, referrer

## **clear Method**

*document.clear*

Closes the document output stream and writes the data to the screen. See the [open](#) method description for more information and examples.

*document*

An object expression that evaluates to a document object.

Not implemented in current build.

### **Applies To**

[Document](#)

### **Methods**

[write](#), [writeLn](#), [open](#), [close](#)

### **Properties**

[linkColor](#), [aLinkColor](#), [vLinkColor](#), [bgColor](#), [fgColor](#), [anchors](#), [links](#), [forms](#), [location](#), [lastModified](#), [title](#), [cookie](#), [referrer](#)

## form Object

An object that resides below the document in the scripting object model. A form may contain:

- Element - The array of objects and intrinsic controls contained in the form.



The form object represents a form in the HTML document. Forms are kept in the document object both as an array and by name. Script forms are accessible either by index (the document's forms array) or by name (given in the NAME="somename" attribute of the HTML <FORM> tag). Given a document with one form defined, the script can access the form in one of two ways:

```
<script language="VBSCRIPT">

' ...first method, by name ...

sub pressedByName
    document.Form1.Button1.value="I've been pressed"    ' access the form
by name
end sub

' ... second method, by index...
' Note that indexes start at 0, not 1!

sub pressedByIndex
    document.form1.elements(1).value="I've been pressed"    ' access the
form by index
end sub
</script>

<form name="Form1">
    <input type="button" name="Button1" value="Press ME"
onClick="pressedByName" language="VBScript">
    <input type="button" name="Button2" value="Press ME"
onClick="pressedByIndex" language="VBScript">
</form>
```

### Methods

submit

### Events

onSubmit

### Properties

action, encoding, method, target, elements, hidden



## **action Property**

*form.action*[=*string*]

Gets or sets the address to be used to carry out the action of the form.

- Returns a string containing the current form action.

*form*

An object expression that evaluates to a form object.

*string*

Optional. A string containing the new action, generally a URL.

If no URL is specified, the base URL of the document is used. Note that this is identical to changing the ACTION attribute of the FORM tag. So the script:

```
document.form[0].action = "http:// www.sample.com/bin/search"
```

is identical to the following:

```
<FORM ACTION="http:// www.sample.com/bin/search">  
</FORM>
```

### **Applies To**

Form

### **Methods**

submit

### **Events**

onSubmit

### **Properties**

encoding, method, target, elements, hidden

## encoding Property

*form.encoding*[=*string*]

Gets or sets the encoding for the form.

- Returns a string containing the current form encoding.

*form*

An object expression that evaluates to a form object.

*string*

Optional. A string containing the new encoding. This must be a valid mime type, like "text/html".

If no mime type is specified, "text/html" is used. Note that this is identical to changing the ENCTYPE attribute of the FORM tag. So the script:

```
document.form[0].action = "http:// www.sample.com/bin/search"  
document.form[0].enctype = "text/html"
```

is identical to the following:

```
<FORM ACTION="http:// www.sample.com/bin/search" ENCTYPE="text/html">  
</FORM>
```

Note that in the current build, encoding can be set, but has no effect on the operation of the form.

### Applies To

Form

### Methods

submit

### Events

onSubmit

### Properties

action, method, target, elements, hidden

## method Property

*form.method*[string]

Indicates how the form data should be sent to the server.

- Returns a string containing the current form method.

*form*

An object expression that evaluates to a form object.

*string*

Optional. A string containing the new method, either GET or POST.

GET means append the arguments to the action URL and open it as if it were an anchor; POST means send the data via an HTTP post transaction. Note that this is identical to the METHOD attribute of the FORM tag, so the script:

```
document.form[0].action = "http:// www.sample.com/bin/search"  
document.form[0].method = "GET"
```

is identical to the following:

```
<FORM ACTION="http:// www.sample.com/bin/search" METHOD=GET>  
</FORM>
```

### Applies To

Form

### Methods

submit

### Events

onSubmit

### Properties

action, encoding, target, elements, hidden

## target Property

*form.target* [=string]

Specifies the name of the target window to display the form results in.

- Returns a string containing the current form target.

*form*

An object expression that evaluates to a form object.

*string*

Optional. A string containing the new target name.

Note that this is identical to the TARGET attribute of the FORM tag, so the script:

```
document.form[0].action = "http:// www.sample.com/bin/search"  
document.form[0].target = "newWindow"
```

is identical to the following:

```
<FORM ACTION="http:// www.sample.com/bin/search" TARGET="newWindow">  
</FORM>
```

Note that in the current build, **target** can be set; however, it has no effect on the operation of the form.

### Applies To

Form

### Methods

submit

### Events

onSubmit

### Properties

action, encoding, method, elements, hidden

## elements Property

*form.elements*[=string]

Returns the array of elements contained in the form.

- Returns an object expression that evaluates to the array of elements in a form.

*form*

An object expression that evaluates to a form object.

The elements include any intrinsics (specified using the INPUT tag) or any embedded objects (specified using the OBJECT tag) contained in the form. So, the HTML:

```
<FORM ACTION="http:// www.sample.com/bin/search" METHOD=GET>
<INPUT NAME="aButton" TYPE ... >
<INPUT NAME="aCheckBox" TYPE ... >
<OBJECT NAME="anObject" DATA=...></OBJECT>
<INPUT NAME="aRadio" TYPE ... >
</FORM>
```

would generate an elements array where *form.elements.length* returns 4, and *form.elements[2].name* returns "anObject".

### Applies To

Form

### Methods

submit

### Events

onSubmit

### Properties

action, encoding, method, target, hidden

## **hidden Property**

Not yet implemented

### **Applies To**

Form

### **Methods**

submit

### **Events**

onSubmit

### **Properties**

action, encoding, method, target, elements

## **submit Method**

*form.submit*

Submits the form. Note that this is identical to clicking a form input with TYPE=SUBMIT.

*form*

An object expression that evaluates to a form object.

### **Applies To**

Form

### **Events**

onSubmit

### **Properties**

action, encoding, method, target, elements, hidden

## onSubmit Event

*form.onSubmit =action*

Fired when the form is submitted.

*form*

An object expression that evaluates to a form object.

*action*

A string expression that evaluates to a scripting function call.

This event can be used to prevent the form from being submitted, or it can be used simply to run additional code before the form is submitted. To prevent the form from being submitted, you must use "return <function>." So, the script:

```
form.onsubmit = "return IsValid()"
```

calls IsValid, and submits the form if it returns TRUE, or doesn't submit the form if it returns FALSE, while:

```
form.onsubmit = "IsValid()"
```

calls IsValid, but submits the form regardless of return value.

In the current build, forms fire the **onSubmit** event when the submit method is called; but not when the submit button is clicked.

### Applies To

Form

### Methods

submit

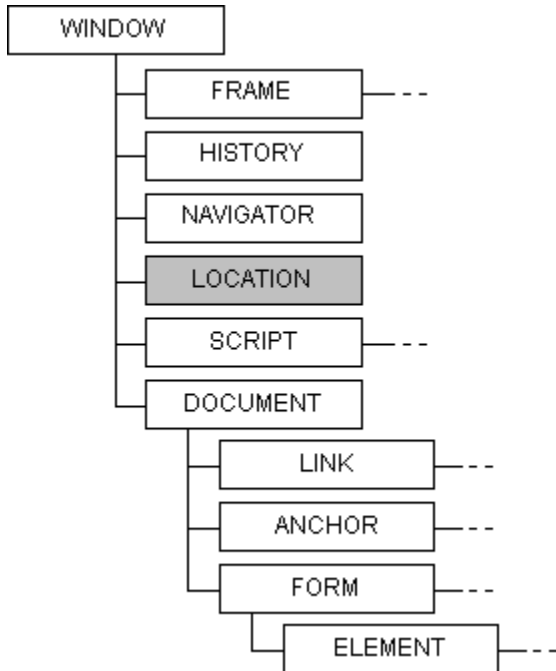
### Properties

action, encoding, method, target, elements, hidden



## location Object

An object that resides below the document in the scripting object model. The location object represents the current URL:



Setting any portion of the location object causes the browser to navigate to the newly constructed URL. The following script navigates to <http://www.microsoft.com>:

```
<script language="VBScript">
    [some preceding VBScript code]
    location.href=" http://www.microsoft.com"
</script>
```

### Properties

[href](#), [protocol](#), [host](#), [hostname](#), [port](#), [pathname](#), [search](#), [hash](#)

## href Property

*location.href* [=string]

Gets or sets the complete URL for the location.

- Returns a string containing the complete URL for the location.

*location*

An object expression that evaluates to a location object.

*string*

Optional. The new string value.

### Applies To

Location

### Properties

protocol, host, hostname, port, pathname, search, hash

## protocol Property

*location.protocol* [=string]

Gets or sets the protocol portion of the URL.

- Returns a string containing the protocol portion of the URL.

*location*

An object expression that evaluates to a location object.

*string*

Optional. The new string value.

For <http://www.microsoft.com>, this would return `http:`.

### Applies To

Location

### Properties

href, host, hostname, port, pathname, search, hash

## host Property

*location*.host [=string]

Gets or sets both the host and port portion of the URL (hostname:port.).

- Returns a string containing the host and port portion of the URL.

*location*

An object expression that evaluates to a location object.

*string*

Optional. The new string value.

For http://www.microsoft.com, this would be www.microsoft.com:80. For file: protocols, this always returns "".

### Applies To

Location

### Properties

href, protocol, hostname, port, pathname, search, hash

## hostname Property

*location.hostname* [=string]

Gets or sets the host portion of the URL, either a name or an IP address.

- Returns a string containing the hostname portion of the URL.

*location*

An object expression that evaluates to a location object.

*string*

Optional. The new string value.

For `http://www.microsoft.com`, this would return `www.microsoft.com`. For file: protocols, this always returns `""`.

### Applies To

Location

### Properties

href, protocol, host, port, pathname, search, hash

## **port Property**

*location.port* [=string]

Gets or sets the port of the URL.

- Returns a string containing the port of the URL.

*location*

An object expression that evaluates to a location object.

*string*

Optional. The new string value.

For `http://www.microsoft.com`, this returns 80. For file: protocols, this always returns "".

### **Applies To**

Location

### **Properties**

href, protocol, host, hostname, pathname, search, hash

## pathname Property

*location.pathname* [=string]

Gets or sets the pathname in the URL.

- Returns a string containing the pathname portion of the URL.  
Note that the current implementation returns "intdev", not "/intdev" as expected.

*location*

An object expression that evaluates to a location object.

*string*

Optional. The new string value.

For <http://www.microsoft.com/intdev>, this returns intdev.

### Applies To

Location

### Properties

href, protocol, host, hostname, port, search, hash

## search Property

*location*.search [=string]

Gets or sets the search portion of the URL, if specified.

- Returns a string containing the search portion of the URL.

*location*

An object expression that evaluates to a location object.

*string*

Optional. The new string value.

For <http://www.microsoft.com/intdev?user>, this returns ?user. For <http://www.microsoft.com/intdev>, this returns NULL.

### Applies To

Location

### Properties

href, protocol, host, hostname, port, pathname, hash



## hash

*location*.hash [=string]

Gets or sets the hash portion of the URL, if specified.

- Returns a string containing the hash portion of the URL.

Note that current implementation returns "#" always.

*location*

An object expression that evaluates to a location object.

*string*

Optional. The new string value.

### **Applies To**

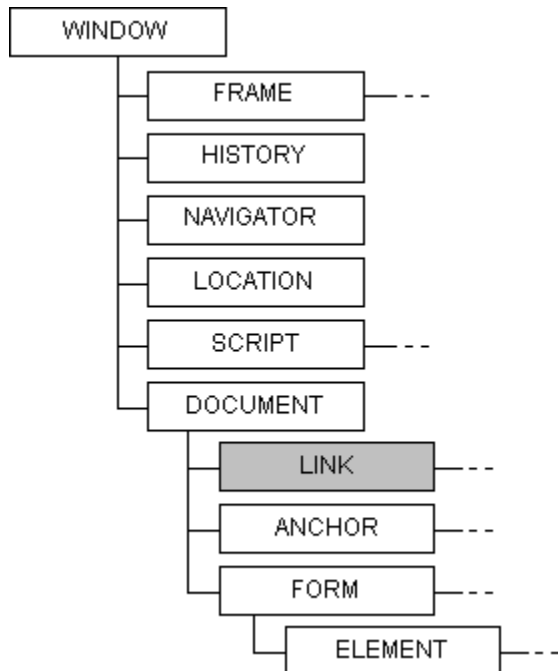
Location

### **Properties**

href, protocol, host, hostname, port, pathname, search

## link Object

An object that resides below the document in the scripting object model. This object specifies an array of links for a given document.



The link object is referenced as a read-only property array. A link object is constructed for every link that appears in the HTML document. A link is defined in scripting as the anchor tag <A> containing the HREF attribute <A HREF="http://www.microsoft.com">. All properties of the link object are read-only and are the same as the location object's properties. It is only accessible through the indexed array. The following lines of script would set linktext to the third link on the page (if it exists):

```
<script language="VBScript">
  [some preceding VBScript code]
  linktext = document.links(2).href
  [some following VBScript code]
</script>
```

### Events

mouseMove, onMouseOver, onClick

### Properties

href, protocol, host, hostname, port, pathname, search, hash, target

## href Property

*link.href*

Returns the complete URL for the link.

- Returns a string containing the complete URL for the link.

*link*

An object expression that evaluates to a link object.

### Applies To

Link

### Events

mouseMove, onMouseOver, onClick

### Properties

protocol, host, hostname, port, pathname, search, hash, target

## **protocol Property**

*link.protocol*

Returns the protocol portion of the URL.

- Returns a string containing the protocol portion of the URL.

*link*

An object expression that evaluates to a link object.

For <http://www.microsoft.com>, this would return `http:`.

### **Applies To**

Link

### **Events**

mouseMove, onMouseOver, onClick

### **Properties**

href, host, hostname, port, pathname, search, hash, target

## host Property

*link*.host

Returns both the host and port portion of the URL (hostname:port).

- Returns a string containing the host and port portion of the URL.

*link*

An object expression that evaluates to a link object.

For <http://www.microsoft.com>, this would return `www.microsoft.com:80`.

### Applies To

Link

### Events

mouseMove, onMouseOver, onClick

### Properties

href, protocol, hostname, port, pathname, search, hash, target

## hostname Property

*link*.hostname

Returns the host portion of the URL, either a name or an IP address.

- Returns a string containing the hostname portion of the URL.

*link*

An object expression that evaluates to a link object.

For <http://www.microsoft.com>, this would return [www.microsoft.com](http://www.microsoft.com).

### Applies To

Link

### Events

mouseMove, onMouseOver, onClick

### Properties

href, protocol, host, port, pathname, search, hash, target

## **port Property**

*link.port*

Returns the port of the URL.

- Returns a string containing the port of the URL.

*link*

An object expression that evaluates to a link object.

For <http://www.microsoft.com>, this returns 80 (the default for HTTP).

### **Applies To**

Link

### **Events**

mouseMove, onMouseOver, onClick

### **Properties**

href, protocol, host, hostname, pathname, search, hash, target

## pathname Property

*link*.pathname

Returns the pathname in the URL.

- Returns a string containing the pathname portion of the URL.

*link*

An object expression that evaluates to a link object.

For <http://www.microsoft.com/intdev>, this returns `/intdev`.

### Applies To

Link

### Events

mouseMove, onMouseOver, onClick

### Properties

href, protocol, host, hostname, port, search, hash, target



## search Property

*link*.search

Returns the search portion of the URL, if specified.

- Returns a string containing the search portion of the URL.  
This returns "user," not "?user," in the current implementation; the leading '?' is omitted.

*link*

An object expression that evaluates to a link object.

For <http://www.microsoft.com/intdev?user>, this returns user.

### Applies To

Link

### Events

mouseMove, onMouseOver, onClick

### Properties

href, protocol, host, hostname, port, pathname, hash, target

## hash Property

*link.hash*

Returns the hash portion of the URL, if specified.

- Returns a string containing the hash portion of the URL.  
This returns NULL in the current implementation when no hash is specified..

*link*

An object expression that evaluates to a link object.

This is the section of the URL after # including the #. For <http://www.microsoft.com/intdev#user>, this returns #user. If no hash is specified, this property returns NULL.

### Applies To

Link

### Events

mouseMove, onMouseOver, onClick

### Properties

href, protocol, host, hostname, port, pathname, search, target

## **target Property**

*link.target*

Returns the target of the link, if specified.

- Returns a string containing the target of the link.

*link*

An object expression that evaluates to a link object.

This is the same as the value of the TARGET attribute of the LINK tag.

### **Applies To**

Link

### **Events**

mouseMove, onMouseOver, onClick

### **Properties**

href, protocol, host, hostname, port, pathname, search, hash

## Events

Link events can be used to set status bar text or other custom actions on mouse movement. The following example is an excerpt from an HTML document that uses a text control to display rich information about the links in an image map. The code decides on the link location.

```
<script language="VBScript" for="Link1" event="mouseMove(shift, button, x,
y)">
  if (InRect(x, y, 5, 30, 120, 85)=true) then
    DescribeLink "A full description of Microsoft's product line"
    [some following VBScript code]
</script>
```

## mouseMove Event

*link.mouseMove* *shift*, *button*, *x*, *y*

Fires an event any time the pointer moves over a link.

*link*

An object expression that evaluates to a link object.

*shift*

The status of the shift key.

*button*

Indicates which button is pressed, if any.

*x*

The horizontal position of the pointer, in pixels.

*y*

The vertical position of the pointer, in pixels.

Shift and button are currently set to zero. x and y contain the actual positional data. To attach scripts or behavior to this event, use the SCRIPT tag as follows:

```
<script language=script-engine for=link-name event="mouseMove(shift, button, x, y)">
```

### Applies To

Link

### Events

onMouseOver, onClick

### Properties

href, protocol, host, hostname, port, pathname, search, hash, target

## **onMouseOver Event**

### ***link.onMouseOver***

Fires an event any time the pointer moves over a link.

#### ***link***

An object expression that evaluates to a link object.

Not implemented in current builds.

To attach scripts or behavior to this event, use the SCRIPT tag as follows:

```
<script language=script-engine for=link-name event="onMouseOver">
```

or attach a script directly in the HTML:

```
<A HREF="http://www.microsoft.com" onMouseOver="alert ('Clicked here')">To  
Microsoft</A>
```

### **Applies To**

Link

### **Events**

mouseMove, onClick

### **Properties**

href, protocol, host, hostname, port, pathname, search, hash, target

## **onClick Event**

### *link.onClick*

Fires an event any time you click on a link.

#### *link*

An object expression that evaluates to a link object.

Not implemented in current builds.

To attach scripts or behavior to this event, use the SCRIPT tag as follows:

```
<script language=script-engine for=link-name event="onClick">
```

or attach a script directly in the HTML:

```
<A HREF="http://www.microsoft.com" onClick="alert ('Clicked here')">To  
Microsoft</A>
```

### **Applies To**

Link

### **Events**

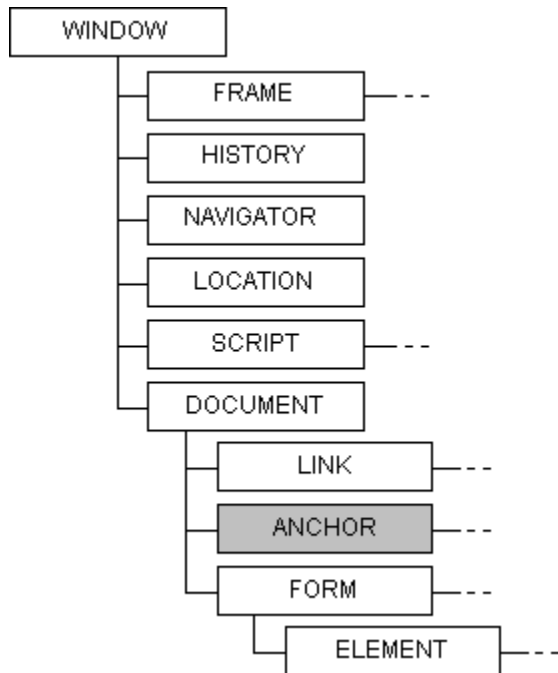
mouseMove, onMouseOver

### **Properties**

href, protocol, host, hostname, port, pathname, search, hash, target

## anchor Object

An object that resides below the document in the scripting object model. This object specifies an array of anchors for a given document. Each entry in this array corresponds to an anchor <A> tag that is found in the corresponding document.



The anchor object is referenced as a read-only property array. An anchor object is constructed for every anchor tag <A> found in the HTML document. It is only accessible through the indexed array. The following lines of script would set anchortext to the name of the third anchor on the page (if it exists).

```
<script language="VBScript">
    [some preceding VBScript code]
    anchortext = document.anchors(2).name
    [some following VBScript code]
</script>
```

### Properties

name



## **name Property**

*anchor.name* [=string]

Gets or sets the name of the anchor.

- Returns a string containing the complete name of the anchor.

*anchor*

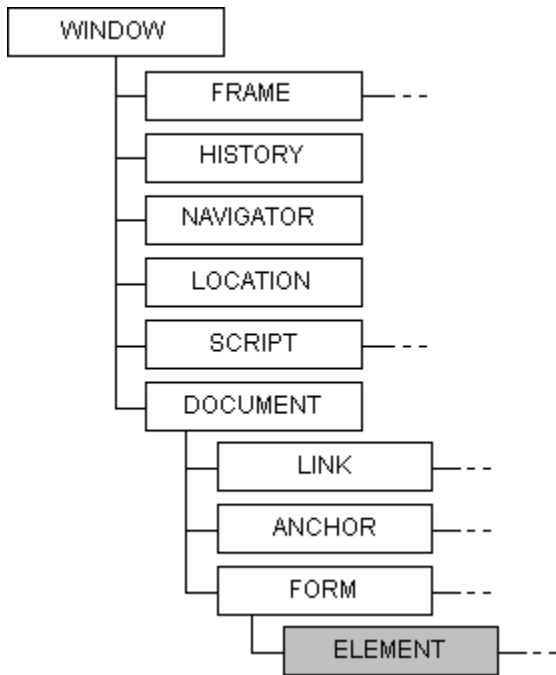
An object expression that evaluates to an anchor object.

*string*

A string containing the new anchor name.

## element Object

An object that resides below the document in the scripting object model. Elements are intrinsic HTML controls or objects. Controls are placed on a document with the <INPUT> tag while objects are placed on a document with the <OBJECT> tag.



Elements are intrinsic HTML controls (placed on a page through the input tag <INPUT>) or objects that are insertable in HTML via the object tag <OBJECT>. These include ActiveX Controls. They can be referenced either by array or name, but this reference must follow the form identifier. Not all properties, methods, and events apply to all elements. Some properties apply to all elements; some only apply to specific elements. See the list below for details by element type, then see the specific method, event, or property documentation for details.

| Element               | Properties  | Methods                                  | Events                     |
|-----------------------|---|--|----------------------------|
| button, reset, submit | form, name, value, enabled  | click, focus                             | onClick, onFocus           |
| check box             | form, name, value, checked, defaultChecked, enabled                 | click, focus                             | onClick, onFocus           |
| radio                 | form, name, value, checked, enabled                                 | click, focus                             | onClick, onFocus           |
| combo                 | form, name, value, enabled, listCount, list, multiSelect, listIndex | click, focus, removeItem, addItem, clear | onClick, onFocus           |
| password              | form, name, value, defaultValue, enabled                            | focus, blur, select                      | onFocus, onBlur            |
| text, text area       | form, name, value,  | focus, blur, select                      | onFocus, onBlur, onChange, |

|        |  |             |                              |
|--------|--|-------------|------------------------------|
|        | defaultValue,<br>enabled                   |             | onSelect                     |
| select | name, length,<br>options,<br>selectedIndex | focus, blur | onFocus, onBlur,<br>onChange |
| hidden | name, value                                |             |                              |

### **Methods**

click, focus, blur, select, removeItem, addItem, clear

### **Events**

onClick, onFocus, onBlur, onChange, onSelect

### **Properties**

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex,  
length, options, selectedIndex

## **form Property**

*element.form*

Gets the form object containing the element.

- An object expression that evaluates to the form containing the element

*element*

Returns an object expression that evaluates to an intrinsic control.

### **Applies To**

**All elements**

### **Methods**

[click](#), [focus](#), [blur](#), [select](#), [removeItem](#), [addItem](#), [clear](#)

### **Events**

[onClick](#), [onFocus](#), [onBlur](#), [onChange](#), [onSelect](#)

### **Properties**

[name](#), [value](#), [defaultValue](#), [checked](#), [defaultChecked](#), [enabled](#), [listCount](#), [multiSelect](#), [listIndex](#), [length](#), [options](#), [selectedIndex](#)

## **name Property**

*element.name* [=string]

Gets or sets the name of the element.

- Returns a string containing the name of the element.

*element*

An object expression that evaluates to an intrinsic control.

*string*

Optional. A string containing the new element name.

### **Applies To**

**All elements**

### **Methods**

[click](#), [focus](#), [blur](#), [select](#), [removeItem](#), [addItemClear](#)

### **Events**

[onClick](#), [onFocus](#), [onBlur](#), [onChange](#), [onSelect](#)

### **Properties**

[form](#), [value](#), [defaultValue](#), [checked](#), [defaultChecked](#), [enabled](#), [listCount](#), [multiSelect](#), [listIndex](#), [length](#), [options](#), [selectedIndex](#)

## value Property

*element.value* [=string]

Gets or sets the value of the element.

- Returns a string containing the value of the element.

*element*

An object expression that evaluates to an intrinsic control.

*string*

Optional. A string containing the new element value.

### Applies To

**All elements**

### Methods

[click](#), [focus](#), [blur](#), [select](#), [removeItem](#), [addItemClear](#)

### Events

[onClick](#), [onFocus](#), [onBlur](#), [onChange](#), [onSelect](#)

### Properties

[form](#), [name](#), [defaultValue](#), [checked](#), [defaultChecked](#), [enabled](#), [listCount](#), [multiSelect](#), [listIndex](#), [length](#), [options](#), [selectedIndex](#)

## **defaultValue Property**

*element.defaultValue* [=string]

Gets or sets the default value of the element.

- Returns a string containing the default value of the element.

*element*

An object expression that evaluates to an intrinsic control.

*string*

Optional. A string containing the new default value.

### **Applies To**

**password, text, text area**

### **Methods**

[click](#), [focus](#), [blur](#), [select](#), [removeItem](#), [addItemClear](#)

### **Events**

[onClick](#), [onFocus](#), [onBlur](#), [onChange](#), [onSelect](#)

### **Properties**

[form](#), [name](#), [value](#), [checked](#), [defaultChecked](#), [enabled](#), [listCount](#), [multiSelect](#), [listIndex](#), [length](#), [options](#), [selectedIndex](#)

## checked Property

*element.checked* [=string]

Gets or sets the checked state of the check box or the radio button.

- Returns TRUE if the check box or radio button is checked; FALSE if not.

*element*

An object expression that evaluates to an intrinsic control.

*string*

Optional. Sets the checked state of the check box or the radio button.

### Applies To

**check box, radio button**

### Methods

[click](#), [focus](#), [blur](#), [select](#), [removeItem](#), [addItem](#), [clear](#)

### Events

[onClick](#), [onFocus](#), [onBlur](#), [onChange](#), [onSelect](#)

### Properties

[form](#), [name](#), [value](#), [defaultValue](#), [defaultChecked](#), [enabled](#), [listCount](#), [multiSelect](#), [listIndex](#), [length](#), [options](#), [selectedIndex](#)



## **defaultChecked Property**

*element.defaultChecked* [=string]

Gets or sets the default checked property of the check box.

- Returns TRUE if the check box is checked by default; FALSE if not.

*element*

An object expression that evaluates to an intrinsic control.

*string*

Optional. Sets the default state of the check box.

### **Applies To**

**check box**

### **Methods**

[click](#), [focus](#), [blur](#), [select](#), [removeItem](#), [addItem](#), [clear](#)

### **Events**

[onClick](#), [onFocus](#), [onBlur](#), [onChange](#), [onSelect](#)

### **Properties**

[form](#), [name](#), [value](#), [defaultValue](#), [checked](#), [enabled](#), [listCount](#), [multiSelect](#), [listIndex](#), [length](#), [options](#), [selectedIndex](#)

## **enabled Property**

*element.enabled* [=bool]

Gets or sets whether the control is enabled.

- Returns TRUE if the control is enabled; FALSE if not.

*element*

An object expression that evaluates to an intrinsic control.

*bool*

Optional. Enables or disables the control.

### **Applies To**

**All elements**

### **Methods**

[click](#), [focus](#), [blur](#), [select](#), [removeItem](#), [addItemClear](#)

### **Events**

[onClick](#), [onFocus](#), [onBlur](#), [onChange](#), [onSelect](#)

### **Properties**

[form](#), [name](#), [value](#), [defaultValue](#), [checked](#), [defaultChecked](#), [listCount](#), [multiSelect](#), [listIndex](#), [length](#), [options](#), [selectedIndex](#)

## **listCount Property**

*element*.listCount

Gets the count of elements in the list.

- Returns the number of elements in the combo box.

*element*

An object expression that evaluates to an intrinsic control.

### **Applies To**

**combo**

### **Methods**

[click](#), [focus](#), [blur](#), [select](#), [removeItem](#), [addItem](#), [clear](#)

### **Events**

[onClick](#), [onFocus](#), [onBlur](#), [onChange](#), [onSelect](#)

### **Properties**

[form](#), [name](#), [value](#), [defaultValue](#), [checked](#), [defaultChecked](#), [enabled](#), [multiSelect](#), [listIndex](#), [length](#), [options](#), [selectedIndex](#)

## **multiSelect Property**

*element*.multiSelect [=bool]

Gets or sets whether the combo is multiselect or not.

- Returns TRUE if the combo is multiselect; FALSE if not.

*element*

An object expression that evaluates to an intrinsic control.

*bool*

Optional. Use TRUE to set the combo to multiselect; FALSE to set to single-select.

### **Applies To**

**combo**

### **Methods**

[click](#), [focus](#), [blur](#), [select](#), [removeItem](#), [addItemClear](#)

### **Events**

[onClick](#), [onFocus](#), [onBlur](#), [onChange](#), [onSelect](#)

### **Properties**

[form](#), [name](#), [value](#), [defaultValue](#), [checked](#), [defaultChecked](#), [enabled](#), [listCount](#), [listIndex](#), [length](#), [options](#), [selectedIndex](#)

## **listIndex Property**

*element*.listIndex [=integer]

Gets or sets the list index.

- Returns the index of the currently selected element. If more than one is selected, it returns the first.

*element*

An object expression that evaluates to an intrinsic control.

*Integer*

Optional. The list index to select. Note that this must be between 0 and ListCount - 1.

**listIndex** is the index of the selected element in the combo. Applies to the combo element.

### **Applies To**

**combo**

### **Methods**

click, focus, blur, select, removeItem, addItem, clear

### **Events**

onClick, onFocus, onBlur, onChange, onSelect

### **Properties**

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, length, options, selectedIndex

## length Property

*element.length*

Gets the number of options in a select element.

- Returns an integer specifying the number of options in a select element.

*element*

An object expression that evaluates to a select element.

### Applies To

**select**

### Methods

[click](#), [focus](#), [blur](#), [select](#), [removeItem](#), [addItem](#), [clear](#)

### Events

[onClick](#), [onFocus](#), [onBlur](#), [onChange](#), [onSelect](#)

### Properties

[form](#), [name](#), [value](#), [defaultValue](#), [checked](#), [defaultChecked](#), [enabled](#), [listCount](#), [multiSelect](#), [listIndex](#), [options](#), [selectedIndex](#)

## options Property

*element.options*

Gets the <options> tags for a select element.

- Returns an array specifying the <options> tag for a select element.

*element*

An object expression that evaluates to a select element.

The options array has the following properties:

- `defaultSelected`  
Identifies the currently selected attribute
- `index`  
Specifies the index of an option
- `length`  
Specifies the number of options in the selected object.
- `name`  
Specifies the name attribute of the selected object.
- `selected`  
Used to programmatically select an option.
- `selectedIndex`  
Specifies the index of the selected option.
- `text`  
Specifies the text to be displayed (this text follows the <option> tag).
- `value`  
Specifies the value attribute.

### Applies To

**select**

### Methods

[click](#), [focus](#), [blur](#), [select](#), [removeItem](#), [addItem](#), [clear](#)

### Events

[onClick](#), [onFocus](#), [onBlur](#), [onChange](#), [onSelect](#)

### Properties

[form](#), [name](#), [value](#), [defaultValue](#), [checked](#), [defaultChecked](#), [enabled](#), [listCount](#), [multiSelect](#), [listIndex](#), [length](#), [selectedIndex](#)

## **selectedIndex Property**

*element.selectedIndex*

Gets the index for the selected option (or the first option selected when there are multiple selected objects).

- Returns an integer specifying the index for the selected option in a select element.

*element*

An object expression that evaluates to a select element.

### **Applies To**

**select**

### **Methods**

[click](#), [focus](#), [blur](#), [select](#), [removeItem](#), [addItemClear](#)

### **Events**

[onClick](#), [onFocus](#), [onBlur](#), [onChange](#), [onSelect](#)

### **Properties**

[form](#), [name](#), [value](#), [defaultValue](#), [checked](#), [defaultChecked](#), [enabled](#), [listCount](#), [multiSelect](#), [listIndex](#), [length](#), [options](#)



## **click Method**

*element.click*

Clicks the element.

*element*

An object expression that evaluates to an intrinsic control.

### **Applies To**

**button, reset, submit, check box, radio, combo**

### **Methods**

focus, blur, select, removeItem, addItem, clear

### **Events**

onClick, onFocus, onBlur, onChange, onSelect

### **Properties**

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

## **focus Method**

*element.focus*

Sets the focus to the element.

*element*

An object expression that evaluates to an intrinsic control.

### **Applies To**

**All elements**

### **Methods**

click, blur, select, removeItem, addItem, clear

### **Events**

onClick, onFocus, onBlur, onChange, onSelect

### **Properties**

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

## **blur Method**

*element.blur*

Clears the focus from the element.

*element*

An object expression that evaluates to an intrinsic control.

### **Applies To**

**password, text, text area**

### **Methods**

click, focus, select, removeItem, addItem, clear

### **Events**

onClick, onFocus, onBlur, onChange, onSelect

### **Properties**

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

## **select Method**

*element.select*

Selects the contents of the element.

*element*

An object expression that evaluates to an intrinsic control.

### **Applies To**

**password, text, text area**

### **Methods**

click, focus, blur, removeItem, addItem, clear

### **Events**

onClick, onFocus, onBlur, onChange, onSelect

### **Properties**

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

## **removeItem Method**

*element.removeItem* *index*

Removes the item at index from the element.

*element*

An object expression that evaluates to an intrinsic control.

*index*

The index of the item to remove. Note that this must be between 0 and ListCount - 1.

### **Applies To**

**combo**

### **Methods**

click, focus, blur, select, addItem, clear

### **Events**

onClick, onFocus, onBlur, onChange, onSelect

### **Properties**

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

## **addItem Method**

*element.addItem index*

Adds the item to the element before the item at index.

*element*

An object expression that evaluates to an intrinsic control.

*index*

The index of the item to add. Note that this must be between 0 and ListCount.

### **Applies To**

**combo**

### **Methods**

click, focus, blur, select, removeItem, clear

### **Events**

onClick, onFocus, onBlur, onChange, onSelect

### **Properties**

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

## **clear Method**

*element.clear index*

Clears the contents of the element.

*element*

An object expression that evaluates to an intrinsic control.

*index*

The index of the item to clear.

### **Applies To**

**combo**

### **Methods**

[click](#), [focus](#), [blur](#), [select](#), [removeItem](#), [addItem](#)

### **Events**

[onClick](#), [onFocus](#), [onBlur](#), [onChange](#), [onSelect](#)

### **Properties**

[form](#), [name](#), [value](#), [defaultValue](#), [checked](#), [defaultChecked](#), [enabled](#), [listCount](#), [multiSelect](#), [listIndex](#), [length](#), [options](#), [selectedIndex](#)

## Events

There are two ways to script events from objects:

- 1 Using the `onEvent="subroutine"` syntax. This method can be used for any HTML intrinsic elements, such as forms, buttons, or links. This method does not work for items inserted using the OBJECT tag. The following example uses this syntax in Button1 to handle `onClick`:

```
<form name="Form1">
  <input type="button" name="Button1" value="Press me" onClick="pressed">
</form>

<script language="VBScript">
  sub pressed
    alert "I've been pressed"
    document.Form1.Button1.value="OUCH"
  end sub
</script>
```

- 2 Using the `FOR="object" EVENT="eventname"` syntax. This method can be used for any named elements, plus any elements inserted using the OBJECT tag. The following example is the same as the first but with a different syntax:

```
<form name="Form1">
  input type="button" name="Button1" value="Press me">
  <script for="Button1" event="onClick" language="VBScript">
    alert "I've been pressed"
    document.Form1.Button1.value="OUCH"
  </script>
</form>
```



## **onClick Event**

*element.onClick*

Fired when the element is clicked.

*element*

An object expression that evaluates to an intrinsic control.

### **Applies To**

**button, reset, submit, check box, radio, combo**

### **Methods**

click, focus, blur, select, removeItem, addItem, clear

### **Events**

onFocus, onBlur, onChange, onSelect

### **Properties**

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

## **onFocus Event**

*element.onFocus*

Fired when the element gets the focus.

*element*

An object expression that evaluates to an intrinsic control.

### **Applies To**

**All elements**

### **Methods**

click, focus, blur, select, removeItem, addItem, clear

### **Events**

onClick, onBlur, onChange, onSelect

### **Properties**

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

## **onBlur Event**

*element.onBlur*

Fired when the element loses the focus.

*element*

An object expression that evaluates to an intrinsic control.

### **Applies To**

**password, text, text area**

### **Methods**

click, focus, blur, select, removeItem, addItem, clear

### **Events**

onClick, onFocus, onChange, onSelect

### **Properties**

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

## **onChange Event**

*element.onChange*

Fired when the element has changed.

*element*

An object expression that evaluates to an intrinsic control.

### **Applies To**

**text, text area**

### **Methods**

click, focus, blur, select, removeItem, addItem, clear

### **Events**

onClick, onFocus, onBlur, onSelect

### **Properties**

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

## **onSelect Event**

*element*.onSelect

Fired when the contents of the element are selected.

*element*

An object expression that evaluates to an intrinsic control.

### **Applies To**

**text, text area**

### **Methods**

click, focus, blur, select, removeItem, addItem, clear

### **Events**

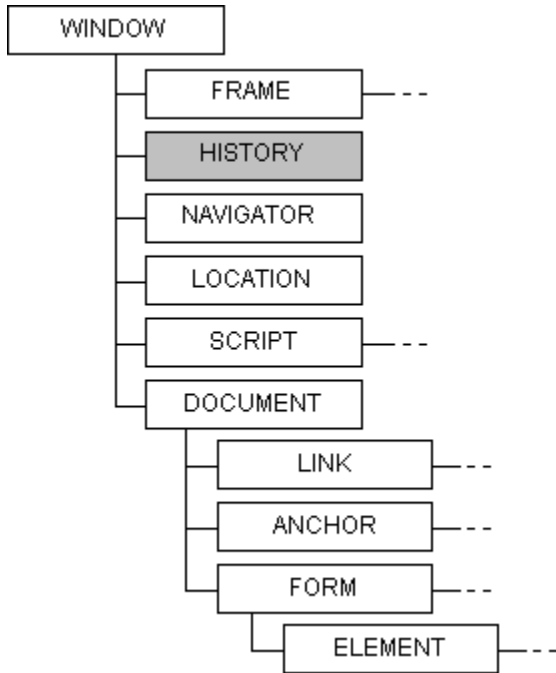
onClick, onFocus, onBlur, onChange

### **Properties**

form, name, value, defaultValue, checked, defaultChecked, enabled, listCount, multiSelect, listIndex, length, options, selectedIndex

## history Object

An object that resides below the window in the scripting object model. This object accesses the history list from the browser.



The history object exposes methods for navigating through the current history.

### Methods

back, forward, go

### Properties

length

## **length Property**

*history.length*

Returns the length of the history list.

- Returns the number of entries in the history.  
Always returns zero in current implementation.

*history*

An object expression that evaluates to a history object.

### **Applies To**

History

### **Methods**

back, forward, go

## **back Method**

*history.back*  $n$

Jumps back in the history  $n$  steps. This behaves exactly as if the user has clicked on the back button  $n$  times.

*history*

An object expression that evaluates to a history object.

$n$

The number of pages to jump back in the history.

Disabled in current implementation.

### **Applies To**

History

### **Methods**

forward, go

### **Properties**

length



## **forward Method**

*history.forward*  $n$

Jumps forward in the history  $n$  steps. This behaves exactly as if the user has clicked on the forward button  $n$  times.

*history*

An object expression that evaluates to a history object.

$n$

The number of pages to jump forward in the history.

Disabled in current implementation.

### **Applies To**

History

### **Methods**

back, go

### **Properties**

length

## **go Method**

*history.go n*

Goes to the  $n$  th item in the history, where *history.go* 1 jumps to the first item and *history.go* *history.length* jumps to the last item.

*history*

An object expression that evaluates to a history object.

$n$

The index of the history entry, from 1 to *history.length*.

Disabled in current implementation.

### **Applies To**

History

### **Methods**

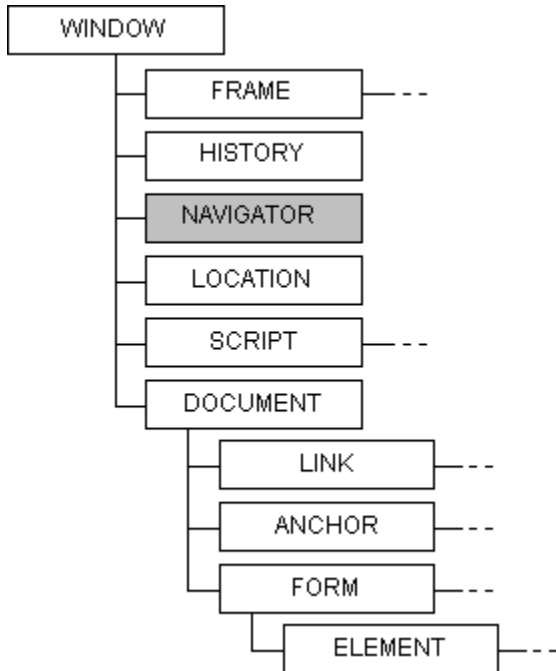
back, forward

### **Properties**

length

## navigator Object

An object that resides below the window in the scripting object model. This object specifies an array of links for a given document.



The navigator object provides information about the browser application to script writers.

### Properties

appCodeName, appName, appVersion, userAgent

## **appName Property**

*navigator.appName*

Returns the code name of the application.

- Returns a string containing the current application code name.

*navigator*

An object expression that evaluates to a history object.

### **Applies To**

Navigator

### **Properties**

appName, appVersion, userAgent

## **appName Property**

*navigator.appName*

Returns the name of the application. Internet Explorer 3.0 currently returns "Microsoft".

- Returns a string containing the current application name.

*navigator*

An object expression that evaluates to a history object.

### **Applies To**

Navigator

### **Properties**

appName, appVersion, userAgent

## **appVersion Property**

*navigator.appVersion*

Returns the version of the application.

- Returns a string containing the current application version.

*navigator*

An object expression that evaluates to a history object.

### **Applies To**

Navigator

### **Properties**

appName, appCodeName, userAgent

## **userAgent Property**

*navigator.userAgent*

Returns the user agent of the application. Internet Explorer 3.0 currently returns "Mozilla/2.0".

- Returns a string containing the current application user agent.

*navigator*

An object expression that evaluates to a history object.

### **Applies To**

Navigator

### **Properties**

appCodeName, appName, appVersion

