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.

If a **CheckBox** is bound to a data source, changing the setting changes the value of that source. 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 use check boxes inside a group box to 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

If a **ComboBox** is bound to a data source, then the **ComboBox** inserts the value the user enters or selects into that data source. If a multicolumn combo box is bound, then the **BoundColumn** property determines which value is stored in the bound data source.

The list in a **ComboBox** consists of rows of data. 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 macro or event procedure assigned to the **CommandButton's** Click event determines what the **CommandButton** does. For example, you can create a **CommandButton** that opens another form. 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.

Frame Control

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3objFrameC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3objFrameX":1} {ewc HLP95EN.DLL,DYNALINK,"Methods":"f3objFrameM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3objFrameS"} {ewc HLP95EN.DLL,DYNALINK,"Properties":"f3objFrameP"} {ewc HLP95EN.DLL,DYNALINK,"Events":"f3objFrameE"}

Creates a functional and visual control group.

Remarks

All option buttons in a **Frame** are mutually exclusive, so you can use the **Frame** to create an option group. You can also use a **Frame** to group controls with closely related contents.For example, in an application that processes customer orders, you might use a **Frame** to group the name, address, and account number of customers.

You can also use a **Frame** to create a group of toggle buttons, but the toggle buttons are not mutually exclusive.

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

Image Control

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3objImageC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3objImageX":1} {ewc HLP95EN.DLL,DYNALINK,"Methods":"f3objImageM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3objImageS"} {ewc HLP95EN.DLL,DYNALINK,"Properties":"f3objImageP"} {ewc HLP95EN.DLL,DYNALINK,"Events":"f3objImageE"}

Displays a picture on a form.

Remarks

The **Image** lets you display a picture as part of the data in a form. For example, you might use an **Image** to display employee photographs in a personnel form.

The **Image** lets you crop, size, or zoom a picture, but does not allow you to edit the contents of the picture. For example, you cannot use the **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.

The **Image** supports the following file formats:

- *.bmp
- *.cur
- *.gif
- *.ico
- *.jpg
- *.wmf

Note You can also display a picture on a **Label**. However, a **Label** does not let you crop, size, or zoom the picture.

The default event for the **Image** 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.

Remarks

If the **ListBox** is bound to a data source, then the **ListBox** stores the selected value in that data source.

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.

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

MultiPage Control

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3objMultiPageC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3objMultiPageX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"f3objMultiPageP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"f3objMultiPageM"} {ewc HLP95EN.DLL,DYNALINK,"Events":"f3objMultiPageE"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3objMultiPageS"}

Presents multiple screens of information as a single set.

Remarks

A **MultiPage** is useful when you work with a large amount of information that can be sorted into several categories. For example, use a **MultiPage** to display information from an employment application. One page might contain personal information such as name and address; another page might list previous employers; a third page might list references. The **MultiPage** lets you visually combine related information, while keeping the entire record readily accessible.

New pages are added to the right of the currently selected page rather than adjacent to it.

Note The **MultiPage** is a container of a **Pages** collection, each of which contains one or more **Page** objects.

The default property for a **MultiPage** is the **Value** property, which returns the index of the currently active **Page** in the **Pages** collection of the **MultiPage**.

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

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. Note that each **OptionButton** in a **Frame** is mutually exclusive.

If an **OptionButton** is bound to a data source, the **OptionButton** can show the value of that data source as either *Yes*/*No*, *True*/*False*, or *On*/*Off*. 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. If the **OptionButton** is bound to a data source, then changing the settings changes the value of that data source. A disabled **OptionButton** is dimmed and does not show a value.

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

You can also use an **OptionButton** inside a group box to 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 a form. 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 form.

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 form.

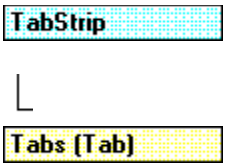
The default property for a **SpinButton** is the **Value** property.

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.



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 or from an organized set of data.

Remarks

A **TextBox** is the control most commonly used to display information entered by a user. Also, it can display a set of data, such as a table, query, worksheet, or a calculation result. If a **TextBox** is bound to a data source, then changing the contents of the **TextBox** also changes the value of the bound data source.

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 a **ToggleButton** is bound to a data source, the **ToggleButton** shows the current value of that data source as either *Yes*/*No*, *True*/*False*, *On*/*Off*, or some other choice of two settings. 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*. If the **ToggleButton** is bound to a data source, changing the setting changes the value of that data source. A disabled **ToggleButton** shows a value, but is dimmed and does not allow changes from the user interface.

You can also use a **ToggleButton** inside a **Frame** to 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**.

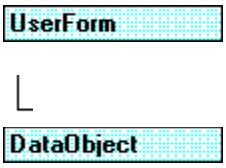
The default property of a **ToggleButton** is the **Value** property.

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

DataObject Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3objDataObjectC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3objDataObjectX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"f3objDataObjectP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"f3objDataObjectM"} {ewc HLP95EN.DLL,DYNALINK,"Events":"f3objDataObjectE"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3objDataObjectS"}

A holding area for formatted text data used in transfer operations. Also holds a list of formats corresponding to the pieces of text stored in the **DataObject**.



Remarks

A **DataObject** can contain one piece of text for the Clipboard text format, and one piece of text for each additional text format, such as custom and user-defined formats.

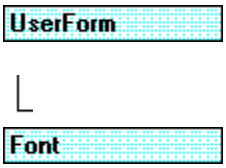
A **DataObject** is distinct from the Clipboard. A **DataObject** supports commands that involve the Clipboard and drag-and-drop actions for text. When you start an operation involving the Clipboard (such as **GetText**) or a drag-and-drop operation, the data involved in that operation is moved to a **DataObject**.

The **DataObject** works like the Clipboard. If you copy a text string to a **DataObject**, the **DataObject** stores the text string. If you copy a second string of the same format to the **DataObject**, the **DataObject** discards the first text string and stores a copy of the second string. It stores one piece of text of a specified format and keeps the text from the most recent operation.

Font Object

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

Defines the characteristics of the text used by a control or form.



Remarks

Each control or form has its own **Font** object to let you set its text characteristics independently of the characteristics defined for other controls and forms. 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 form or container determine the default font attributes of controls you put on the form.

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"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3objLabelX":1} {ewc HLP95EN.DLL,DYNALINK,"Methods":"f3objLabelM"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3objLabelS"} {ewc HLP95EN.DLL,DYNALINK,"Properties":"f3objLabelP"} {ewc HLP95EN.DLL,DYNALINK,"Events":"f3objLabelE"}

Displays descriptive text.

Remarks

A **Label** control on a form 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 display values from data sources or expressions; it is always unbound and doesn't change as you move from record to record.

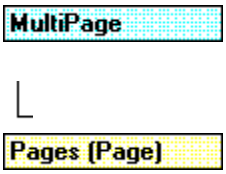
The default property for a **Label** is the **Caption** property.

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

Page Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3objPageC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3objPageX":1}
{ewc HLP95EN.DLL,DYNALINK,"Properties":"f3objPageP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"f3objPageM"}
{ewc HLP95EN.DLL,DYNALINK,"Events":"f3objPageE"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3objPageS"}

One page of a **MultiPage** and a single member of a **Pages** collection.



Remarks

Each **Page** object contains its own set of controls and does not necessarily rely on other pages in the collection for information. A **Page** inherits some properties from its container; the value of each inherited property is set by the container.

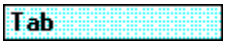
A **Page** has a unique name and index value within a **Pages** collection. You can reference a **Page** by either its name or its index value. The index of the first **Page** in a collection is 0; the index of the second **Page** is 1; and so on. When two **Page** objects have the same name, you must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

The default name for the first **Page** is Page1; the default name for the second **Page** is Page2.

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.



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 a **Page**, a **Tab** does not contain any controls. Controls that appear within the region bounded by a **TabStrip** are contained on the form, 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.

Controls Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3objControlsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3objControlsX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3objControlsA"} {ewc HLP95EN.DLL,DYNALINK,"Properties":"f3objControlsP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"f3objControlsM"} {ewc HLP95EN.DLL,DYNALINK,"Events":"f3objControlsE"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3objControlsS"}

Includes all the controls contained in an object.

Controls

Remarks

Each control in the **Controls** collection of an object has a unique index whose value can be either an integer or a string. The index value for the first control in a collection is 0; the value for the second control is 1, and so on. This value indicates the order in which controls were added to the collection.

If the index is a string, it represents the name of the control. The **Name** property of a control also specifies a control's name.

You can use the **Controls** collection to enumerate or count individual controls, and to set their properties. For example, you can enumerate the **Controls** collection of a particular form and set the **Height** property of each control to a specified value.

Note The For Each...Next statement is useful for enumerating a collection.

Pages Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3objPageCollC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3objPageCollX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3objPageCollA"} {ewc HLP95EN.DLL,DYNALINK,"Properties":"f3objPageCollP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"f3objPageCollM"} {ewc HLP95EN.DLL,DYNALINK,"Events":"f3objPageCollE"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3objPageCollS"}

A **Pages** collection includes all the pages of a **MultiPage**.

Remarks

Each **Pages** collection provides the features to manage the number of pages in the collection and to identify the page that is currently in use.

A **Page** object has a unique name and index value within a **Pages** collection. You can reference a **Page** either by its name or its index value. The index of the first **Page** in a collection is 0; the index of the second **Page** is 1; and so on. The default name for the first page is Page1; the default name for the second page is Page2.

The default value of a **Pages** collection identifies the current **Page** of a collection.

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**.

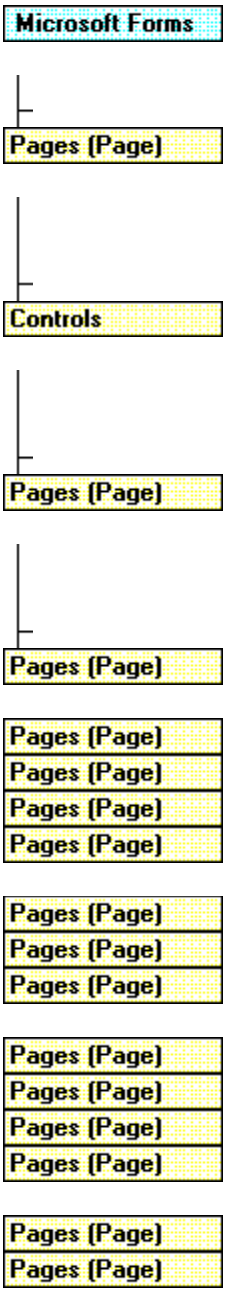
Remarks

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.

Microsoft Forms Object Model Overview



UserForm Window

AddControl Event

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3evtAddControlC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3evtAddControlX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3evtAddControlA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3evtAddControlS"}

Occurs when a control is inserted onto a form, a **Frame**, or a **Page** of a **MultiPage**.

Syntax

For Frame
Private Sub *object*_**AddControl( )**
For MultiPage
Private Sub *object*_**AddControl(** *index* **As Long**, *ctrl* **As Control)**

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

Part	Description
object	Required. A valid object.
index	Required. The index of the **Page** that will contain the new control.
ctrl	Required. The control to be added.

Remarks

The AddControl event occurs when a control is added at run time. This event is not initiated when you add a control at design time, nor is it initiated when a form is initially loaded and displayed at run time.

The default action of this event is to add a control to the specified form, **Frame**, or **MultiPage**.

The **Add** method initiates the AddControl event.

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
object	Required. A valid object.

Remarks

The AfterUpdate event occurs regardless of whether the control is bound (that is, when the **RowSource** property specifies a data source for the control). 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 Frame

Private Sub *object*_**BeforeDragOver(ByVal** *Cancel* **As MSForms.ReturnBoolean**, *ctrl* **As Control**, **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 MultiPage

Private Sub *object*_**BeforeDragOver(** *index* **As Long**, **ByVal** *Cancel* **As MSForms.ReturnBoolean**, *ctrl* **As Control**, **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 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
object	Required. A valid object name.
index	Required. The index of the **Page** in a **MultiPage** that the drag-and-drop operation will affect.
Cancel	Required. Event status. **False** indicates that the control should handle the event (default). **True** indicates the application handles the event.
ctrl	Required. The control being dragged over.
Data	Required. Data that is dragged in a drag-and-drop operation. The data is packaged in a **DataObject**.
X, Y	Required. The horizontal and vertical coordinates of the control's position. Both coordinates are measured in points. *X* is measured from the left edge of the control; *Y* is measured from the top of the control..
DragState	Required. Transition state of the data being dragged.
Effect	Required. Operations supported by the drop source.
Shift	Required. Specifies the state of SHIFT, CTRL, and ALT.

Settings

The settings for *DragState* are:

Constant	Value	Description
fmDragStateEnter	0	Mouse pointer is within range of a target.
fmDragStateLeave	1	Mouse pointer is outside the range of a

		target.
fmDragStateOver	2	Mouse pointer is at a new position, but remains within range of the same target.

The settings for *Effect* are:

Constant	Value	Description
fmDropEffectNone	0	Does not copy or move the drop source to the drop target.
fmDropEffectCopy	1	Copies the drop source to the drop target.
fmDropEffectMove	2	Moves the drop source to the drop target.
fmDropEffectCopyOrMove	3	Copies or moves the drop source to the drop target.

The settings for *Shift* are:

Constant	Value	Description
fmShiftMask	1	SHIFT was pressed.
fmCtrlMask	2	CTRL was pressed.
fmAltMask	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 Frame

Private Sub *object*_**BeforeDropOrPaste(ByVal** *Cancel* **As MSForms.ReturnBoolean**, *ctrl* **As Control**, **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 MultiPage

Private Sub *object*_**BeforeDropOrPaste(** *index* **As Long**, **ByVal** *Cancel* **As MSForms.ReturnBoolean**, *ctrl* **As Control**, **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 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
object	Required. A valid object name.
index	Required. The index of the **Page** in a **MultiPage** that the drop or paste operation will affect.
Cancel	Required. Event status. **False** indicates that the control should handle the event (default). **True** indicates the application handles the event.
ctrl	Required. The target control.
Action	Required. Indicates the result, based on the current keyboard settings, of the pending drag-and-drop operation.
Data	Required. Data that is dragged in a drag-and-drop operation. The data is packaged in a **DataObject**.
X, Y	Required. The horizontal and vertical position of the mouse pointer when the drop occurs. Both coordinates are measured in points. *X* is measured from the left edge of the control; *Y* is measured from the top of the control..
Effect	Required. Effect of the drag-and-drop operation on the target control.
Shift	Required. Specifies the state of SHIFT, CTRL, and ALT.

Settings

The settings for *Action* are:

Constant	Value	Description

fmActionPaste	2	Pastes the selected object into the drop target.
fmActionDragDrop	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
fmDropEffectNone	0	Does not copy or move the drop source to the drop target.
fmDropEffectCopy	1	Copies the drop source to the drop target.
fmDropEffectMove	2	Moves the drop source to the drop target.
fmDropEffectCopyOrMove	3	Copies or moves the drop source to the drop target.

The settings for *Shift* are:

Constant	Value	Description
fmShiftMask	1	SHIFT was pressed.
fmCtrlMask	2	CTRL was pressed.
fmAltMask	4	ALT was pressed.

Remarks

For a **MultiPage** or **TabStrip**, Visual Basic for Applications 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(ByVal** *Cancel* **As MSForms.ReturnBoolean)**

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

Part	Description
object	Required. A valid object.
Cancel	Required. Event status. **False** indicates that the control should handle the event (default). **True** cancels the update and indicates the application should handle the event.

Remarks

The BeforeUpdate event occurs regardless of whether the control is bound (that is, when the **RowSource** property specifies a data source for the control). This event occurs before the AfterUpdate and Exit events for the control (and before the Enter event for the next control that receives 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
object	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**.
- Selecting a different page on a **MultiPage**.

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

Syntax

For MultiPage, TabStrip
Private Sub *object*_**Click(** *index* **As Long)**
For all other controls
Private Sub *object*_**Click( )**

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

Part	Description
object	Required. A valid object.
index	Required. The index of the page or tab in a **MultiPage** or **TabStrip** associated with this event.

Remarks

Of the two cases where the Click event occurs, the first case applies to the **CommandButton**, **Frame**, **Image**, **Label**, **ScrollBar**, and **SpinButton**. The second case applies to the **CheckBox**, **ComboBox**, **ListBox**, **MultiPage**, **TabStrip**, and **ToggleButton**. It also applies to an **OptionButton** when the value changes to **True**.

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

- Clicking a blank area of a form or a disabled control (other than a list box) on the form.
- 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 ENTER on a form that has a command button whose **Default** property is set to **True**, as long as no other command button has the focus.
- Pressing ESC on a form that has a command button whose **Cancel** property is set to **True**, as long as no other command button has the focus.
- 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 different page or tab in a **MultiPage** or **TabStrip**. The **Value** property for these controls reflects the current **Page** or **Tab**. Clicking the current page or tab does not change the control's value and does not initiate the Click event.
- 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.

DblClick Event

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3evtDblClickC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3evtDblClickX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3evtDblClickA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3evtDblClickS"}

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

Syntax

For MultiPage, TabStrip

Private Sub *object*_**DblClick(** *index* **As Long**, **ByVal** *Cancel* **As MSForms.ReturnBoolean)**

For other controls

Private Sub *object*_**DblClick(ByVal** *Cancel* **As MSForms.ReturnBoolean)**

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

Part	Description
object	Required. A valid object.
index	Required. The position of a **Page** or **Tab** object within a **Pages** or **Tabs** collection.
Cancel	Required. Event status. **False** indicates that the control should handle the event (default). **True** 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 DblClick event:

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

If a control, such as **TextBox**, does not support Click, Click is omitted fom the order of events leading to the DblClick 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 form 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
object	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 form. Exit occurs immediately before a control loses the focus to another control on the same form.

Syntax

Private Sub *object*_**Enter( )**
Private Sub *object*_**Exit(ByVal** *Cancel* **As MSForms.ReturnBoolean)**

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

Part	Description
object	Required. A valid object name.
Cancel	Required. Event status. **False** indicates that the control should handle the event (default). **True** 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 Visual Basic. Unlike GotFocus and LostFocus, the Enter and Exit events don't occur when a form 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 form, the Exit event is initiated for the check box (because focus is moving to a different object in the same form) and then the Enter event occurs for the second control on the form.

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 a macro or event procedure to display a small form 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

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

Syntax

For MultiPage

Private Sub *object*_**Error(** *index* **As Long**, **ByVal** *Number* **As Integer**, **ByVal** *Description* **As MSForms.ReturnString**, **ByVal** *SCode* **As SCode**, **ByVal** *Source* **As String**, **ByVal** *HelpFile* **As String**, **ByVal** *HelpContext* **As Long**, **ByVal** *CancelDisplay* **As MSForms.ReturnBoolean)**

For other controls

Private Sub *object*_**Error(ByVal** *Number* **As Integer**, **ByVal** *Description* **As MSForms.ReturnString**, **ByVal** *SCode* **As SCode**, **ByVal** *Source* **As String**, **ByVal** *HelpFile* **As String**, **ByVal** *HelpContext* **As Long**, **ByVal** *CancelDisplay* **As MSForms.ReturnBoolean)**

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

Part	Description
object	Required. A valid object name.
index	Required. The index of the page in a **MultiPage** associated with this event.
Number	Required. Specifies a unique value that the control uses to identify the error.
Description	Required. A textual description of the error.
SCode	Required. Specifies the OLE status code for the error. The low-order 16 bits specify a value that is identical to the *Number* argument.
Source	Required. The string that identifies the control which initiated the event.
HelpFile	Required. Specifies a fully qualified path name for the Help file that describes the error.
HelpContext	Required. Specifies the context ID of the Help file topic that contains a description of the error.
CancelDisplay	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(ByVal** *KeyCode* **As MSForms.ReturnInteger**, **ByVal** *Shift* **As fmShiftState)**
Private Sub *object*_**KeyUp(ByVal** *KeyCode* **As MSForms.ReturnInteger**, **ByVal** *Shift* **As fmShiftState)**

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

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

Settings

The settings for *Shift* are:

Constant	Value	Description
fmShiftMask	1	SHIFT was pressed.
fmCtrlMask	2	CTRL was pressed.
fmAltMask	4	ALT was pressed.

Remarks

The KeyDown event occurs when the user presses a key on a running form while that form 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 form or control with the focus receives all keystrokes. A form 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 a form or control using either the SendKeys action in a macro or the SendKeys Statement in Visual Basic.

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 user presses enter on a form with a command button whose **Default** property is set to **True**.
- The user presses esc on a form with a command button whose **Cancel** property is set to **True**.

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 forms and controls on a form. 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(ByVal** *KeyANSI* **As MSForms.ReturnInteger)**

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

Part	Description
object	Required. A valid object.
KeyANSI	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 form while the form 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 a form or control using either the SendKeys action in a macro or the SendKeys Statement in Visual Basic.

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 form or control with the focus receives all keystrokes. A form 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

Layout Event

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3evtLayoutC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3evtLayoutX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3evtLayoutA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3evtLayoutS"}

Occurs when a form, **Frame**, or **MultiPage** changes size.

Syntax

For MultiPage
Private Sub *object*_**Layout(** *index* **As Long)**
For all other controls
Private Sub *object*_**Layout( )**

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

Part	Description
object	Required. A valid object.
index	Required. The index of the page in a **MultiPage** that changed size.

Remarks

The default action of the layout event is to calculate new positions of controls and to repaint the screen.

A user can initiate the Layout event by changing the size of a control.

For controls that support the **AutoSize** property, the Layout event is initiated when **AutoSize** changes the size of the control. This occurs when the user changes the value of a property that affects the size of a control. For example, increasing the **Font** size of a **TextBox** or **Label** can significantly change the dimensions of the control and initiate a Layout event.

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 MultiPage, 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
object	Required. A valid object.
index	Required. The index of the page or tab in a **MultiPage** or **TabStrip** with the specified event.
Button	Required. An integer value that identifies which mouse button caused the event.
Shift	Required. The state of SHIFT, CTRL, and ALT.
X, Y	Required. The horizontal or vertical position, in points, from the left or top edge of the form, **Frame**, or **Page**.

Settings

The settings for *Button* are:

Constant	Value	Description
fmButtonLeft	1	The left button was pressed.
fmButtonRight	2	The right button was pressed.
fmButtonMiddle	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
fmShiftMask	1	Mask to detect SHIFT.
fmCtrlMask	2	Mask to detect CTRL.
fmAltMask	4	Mask to detect ALT.

Remarks

For a **MultiPage**, the MouseDown event occurs when the user presses a mouse button over the control.

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 a form, 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 form.

The sequence of mouse-related events is:

1. MouseDown
2. MouseUp
3. Click
4. DblClick
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 a form 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 MultiPage, 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
object	Required. A valid object name.
index	Required. The index of the page or tab in a **MultiPage** or **TabStrip** associated with this event.
Button	Required. An integer value that identifies the state of the mouse buttons.
Shift	Required. Specifies the state of SHIFT, CTRL, and ALT.
X, Y	Required. The horizontal or vertical position, measured in points, from the left or top edge of the control.

Settings

The *index* argument specifies which page or tab was clicked over. A –1 designates that the user did not click on any of the pages or 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:

Constant	Value	Description
fmShiftMask	1	Mask to detect SHIFT.
fmCtrlMask	2	Mask to detect CTRL.
fmAltMask	4	Mask to detect ALT.

Remarks

The MouseMove event applies to forms, controls on a form, 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 a form can also generate a MouseMove event even if the mouse is stationary. MouseMove events are generated when the form moves underneath the pointer. If a macro or event procedure moves a form 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.

RemoveControl Event

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3evtRemoveControlC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3evtRemoveControlX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3evtRemoveControlA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3evtRemoveControlS"}

Occurs when a control is deleted from the container.

Syntax

For MultiPage

Private Sub *object*_**RemoveControl(** *index* **As Long**, *ctrl* **As Control)**

For all other controls

Private Sub *object*_**RemoveControl(** *ctrl* **As Control)**

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

Part	Description
object	Required. A valid object name.
index	Required. The index of the page in a **MultiPage** that contained the deleted control.
ctrl	Required. The deleted control.

Remarks

This event occurs when a control is deleted from the form, not when a control is unloaded due to a form being closed.

Scroll Event

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

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**, **ByVal** *ActualDx* **As MSForms.ReturnSingle**, **ByVal** *ActualDy* **As MSForms.ReturnSingle)**

For Frame

Private Sub *object*_**Scroll(** *ActionX* **As fmScrollAction**, *ActionY* **As fmScrollAction**, **ByVal** *RequestDx* **As Single**, **ByVal** *RequestDy* **As Single**, **ByVal** *ActualDx* **As MSForms.ReturnSingle**, **ByVal** *ActualDy* **As MSForms.ReturnSingle)**

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

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

Settings

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

Constant	Value	Description
fmScrollActionNoChange	0	No change occurred.
fmScrollActionLineUp	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.
fmScrollActionLineDown	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.
fmScrollActionPageUp	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.
fmScrollActionPageDown	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.
fmScrollActionBegin	5	The top of a vertical scroll bar; the left end of a horizontal scroll bar.
fmScrollActionEnd	6	The bottom of a vertical scroll bar; the right end of a horizontal scroll bar.
fmScrollActionPropertyChange	8	The value of either the **ScrollTop** or the **ScrollLeft** property changed. The direction and amount of movement depend on which property was changed and on the new property value.
fmScrollActionControlRequest	9	A control asked its container to scroll. The amount of movement depends on the specific control and container involved.
fmScrollActionFocusRequest	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:

Part	Description
object	Required. A valid object.

Remarks

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

Zoom Event

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3evtZoomC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3evtZoomX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3evtZoomS"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3evtZoomA"}

Occurs when the value of the **Zoom** property changes.

Syntax

For Frame
Private Sub *object*_**Zoom(** *Percent* **As Integer)**
For MultiPage
Private Sub *object*_**Zoom(** *index* **As Long**, *Percent* **As Integer)**

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

Part	Description
object	Required. A valid object name.
index	Required. The index of the page in a **MultiPage** associated with this event.
Percent	Required. The percentage the form is to be zoomed. Valid values range from 10 percent to 400 percent.

Remarks

The value of the **Zoom** property identifies how the size of the form or **Page** changes. The value of the property indicates how the size of the control should change relative to its current size. Values less than 100 reduce the displayed size of the form; values greater than 100 increase the displayed size of the form.

You can set this property to any integer from 10 to 400.

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** or **Page** in a **TabStrip** or **MultiPage**, or adds a control by its programmatic identifier (*ProgID*) to a page or form.

Syntax

For MultiPage, 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
object	Required. A valid object name.
Name	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 form is used.
Caption	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 form is used.
index	Optional. Identifies the position of a page or tab within a **Pages** or **Tabs** collection. If an index is not specified, the system appends the page or tab to the end of the **Pages** or **Tabs** collection and assigns the appropriate index value.
ProgID	Required. Programmatic identifier. A text string with no spaces that identifies an object class. The standard syntax for a *ProgID* is <Vendor>.<Component>.<Version>. A *ProgID* is mapped to a class identifier (CLSID).
Visible	Optional. **True** if the object is visible (default). **False** if the object is hidden.

Settings

ProgID values for individual controls are:

CheckBox	Forms.CheckBox.1
ComboBox	Forms.ComboBox.1
CommandButton	Forms.CommandButton.1
Frame	Forms.Frame.1
Image	Forms.Image.1
Label	Forms.Label.1
ListBox	Forms.ListBox.1
MultiPage	Forms.MultiPage.1
OptionButton	Forms.OptionButton.1
ScrollBar	Forms.ScrollBar.1
SpinButton	Forms.SpinButton.1
TabStrip	Forms.TabStrip.1
TextBox	Forms.TextBox.1

ToggleButton	Forms.ToggleButton.1

Remarks

For a **MultiPage** control, the **Add** method returns a **Page** object. For a **TabStrip**, it returns a **Tab** object. The index value for the first **Page** or **Tab** of a collection is 0, the value for the second **Page** or **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 AddControl event occurs after the control is added.

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

```
userform1.thebox.text
```

If you add a control at 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 **Name** 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
object	Required. A valid object.
Item	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.
varIndex	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.

If the control is bound to data, the **AddItem** method fails.

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,"Specifics":"f3mthClearS"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3mthClearA"}

Removes all objects from an object or collection.

Syntax

object.**Clear**

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

Part	Description
object	Required. A valid object.

Remarks

For a **MultiPage** or **TabStrip**, the **Clear** method deletes individual pages or 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.

If the control is bound to data, the **Clear** method fails.

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
object	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 **Page**, the **Copy** method copies the currently selected control or controls. On a **TextBox** or **ComboBox**, it copies the currently selected text.

Using **Copy** for a form, **Frame**, or **Page** 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
object	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 a **Page**, **Frame**, or form, **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**.

Syntax

object.**DropDown**

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

Part	Description
object	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
object	Required. A valid object.
format	Required. An integer or string specifying a specific format that might exist in the **DataObject**. If the specified format exists in the **DataObject**, **GetFormat** returns **True**.

Settings

The settings for *format* are:

Value	Description
1	Text format.
A string or any integer other than 1	A user-defined **DataObject** format passed to the **DataObject** from **SetText**.

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 **True**; if not, **GetFormat** returns **False**.

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

GetFromClipboard Method

{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"}

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

Syntax

String = *object.***GetFromClipboard( )**

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

Part	Description
object	Required. A valid object name.

Remarks

The **DataObject** can contain multiple data items, but each item must be in a different format. For example, the **DataObject** might include one text item and one item in a custom format; but cannot include two text items.

GetText Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3mthGetTextC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3mthGetTextX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3mthGetTextA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3mthGetTextS"}

Retrieves a text string from the **DataObject** using the specified format.

Syntax

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

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

Part	Description
object	Required. A valid object name.
format	Optional. A string or integer specifying the format of the data to retrieve from the **DataObject**.

Settings

The settings for *format* are:

Value	Description
1	Text format.
A string or any integer other than 1	A user-defined **DataObject** format passed to the **DataObject** from **SetText**.

Remarks

The **DataObject** supports multiple formats, but only supports one data item of each format. For example, the **DataObject** might include one text item and one item in a custom format; but cannot include two text items.

If no format is specified, the **GetText** method requests information in the Text format from the **DataObject**.

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:

Part	Description
object	Required. A valid object.
collectionindex	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 a form or control, or moves all the controls in the **Controls** collection..

Syntax

For a form or control

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

For the Controls collection

object.**Move(** X, Y**)**

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

Part	Description
object	Required. A valid object name.
Left	Optional. Single-precision value, in points, indicating the horizontal coordinate for the left edge of the object.
Top	Optional. Single-precision value, in points, that specifies the vertical coordinate for the top edge of the object.
Width	Optional. Single-precision value, in points, indicating the width of the object.
Height	Optional. Single-precision value, in points, indicating the height of the object.
Layout	Optional. A Boolean value indicating whether the Layout event is initiated for the control's parent following this move. **False** is the default value.
X, Y	Required. Single-precision value, in points, that specifies the change from the current horizontal and vertical position for each control in the **Controls** collection.

Settings

The maximum and minimum values for the *Left*, *Top*, *Width*, *Height*, *X*, and *Y* arguments vary from one application to another.

Remarks

For a form or control, you can move a selection to a specific location relative to the edges of the form 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.

For the **Controls** collection, you can move all the controls in this collection a specific distance from their current positions on a form, **Frame**, or **Page**.

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
object	Required. A valid object.

Remarks

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

When the paste method is used with a form, you can paste any object onto the form.

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
object	Required. A valid object.

Remarks

The **PutInClipboard** method replaces the contents of the Clipboard with the contents of the **DataObject** that is in Text format.

RedoAction Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3mthRedoActionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3mthRedoActionX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3mthRedoActionA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3mthRedoActionS"}

Reverses the effect of the most recent Undo action.

Syntax

Boolean = *object*.**RedoAction**

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

Part	Description
object	Required. A valid object.

Remarks

Redo reverses the last Undo, which is not necessarily the last action taken. Not all actions can be undone.

For example, after pasting text into a **TextBox** and then choosing the Undo command to remove the text, you can choose the Redo command to put the text back in.

Note If the **CanRedo** property is **False**, the Redo command is not available in the user interface, and the **RedoAction** method is not valid in code.

RedoAction returns **True** if it was successful.

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 a **Frame**, **Page**, or form.

Syntax

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

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

Part	Description
object	Required. A valid object.
collectionindex	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
object	Required. A valid object.
index	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.

This method will not remove a row from the list if the **ListBox** is data bound (that is, when the **RowSource** property specifies a data source for the **ListBox**).

Repaint Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3mthRepaintC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3mthRepaintX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3mthRepaintA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3mthRepaintS"}

Updates the display by redrawing the form or page.

Syntax

Boolean = *object*.**Repaint**

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

Part	Description
object	Required. A valid object.

Remarks

The **Repaint** method is useful if the contents or appearance of an object changes significantly, and you don't want to wait until the system automatically repaints the area.

Scroll Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3mthScrollC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3mthScrollX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3mthScrollS"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3mthScrollA"}

Moves the scroll bar on an object.

Syntax

object.**Scroll(** [*ActionX* [, *ActionY*]]**)**

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

Part	Description
object	Required. A valid object name.
ActionX	Optional. Identifies the action to occur in the horizontal direction.
ActionY	Optional. Identifies the action to occur in the vertical direction.

Settings

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

Constant	Value	Description
fmScrollActionNoChange	0	Do not scroll in the specified direction.
fmScrollActionLineUp	1	Move up on a vertical scroll bar or left on a horizontal scroll bar. Movement is equivalent to pressing the up or left arrow key on the keyboard to move the scroll bar.
fmScrollActionLineDown	2	Move down on a vertical scroll bar or right on a horizontal scroll bar. Movement is equivalent to pressing the right or down arrow key on the keyboard to move the scroll bar.
fmScrollActionPageUp	3	Move one pageup on a vertical scroll bar or one page left on a horizontal scroll bar. Movement is equivalent to pressing PAGE UP on the keyboard to move the scroll bar.
fmScrollActionPageDown	4	Move one pagedown on a vertical scroll bar or one page right on a horizontal scroll bar. Movement is equivalent to pressing PAGE DOWN on the keyboard to move the scroll bar.
fmScrollActionBegin	5	Move to the top of a vertical scroll bar or to the left end of a horizontal scroll bar.
fmScrollActionEnd	6	Move to the bottom of a vertical scroll bar or to the right end of a

horizontal scroll bar.

Remarks

The **Scroll** method applies scroll bars that appear on a form, **Frame**, or **Page** that is larger than its display area. This method does not apply to the stand-alone **ScrollBar** or to scroll bars that appear on a **TextBox**.

SetDefaultTabOrder Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3mthSetDefaultTabOrderC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3mthSetDefaultTabOrderX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3mthSetDefaultTabOrderA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3mthSetDefaultTabOrderS"}

Sets the **TabIndex** property of each control on a form, using a default top-to-bottom, left-to-right tab order.

Syntax

object.**SetDefaultTabOrder**

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

Part	Description
object	Required. A valid object.

Remarks

Microsoft Forms sets the tab order beginning with controls in the upper left corner of the form and moving to the right. It places controls closest to the left edge of the form earlier in the tab order. If more than one control is the same distance from the left edge of the form, tab order values are assigned from top to bottom.

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
object	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.

The **SetFocus** method is valid for an empty **Frame** as well as a **Frame** that contains other controls. An empty **Frame** will take the focus itself, and any subsequent keyboard events apply to the **Frame**. In a **Frame** that contains other controls, the focus moves to the first control in the **Frame**, and subsequent keyboard events apply to the control that has the focus.

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 **DataObject** using a specified format.

Syntax

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

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

Part	Description
object	Required. A valid object.
StoreData	Required. Defines the data to store on the **DataObject**.
format	Optional. An integer or string specifying the format of *StoreData*. When retrieving data from the **DataObject**, the format identifies the piece of data to retrieve.

Settings

The settings for *format* are:

Value	Description
1	Text format.
A string or integer value other than 1	A user-defined **DataObject** format.

Remarks

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

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**, and the previously existing data is there as well.

If no format is specified, the **SetText** method assigns the Text format to the text string. If a new format is specified, the **DataObject** 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
Object	Required. A valid object.
Effect	Optional. Effect of the drop operation on the target control.

Settings

The settings for *Effect* are:

Constant	Value	Description
fmDropEffectNone	0	Does not copy or move the drop source to the drop target.
fmDropEffectCopy	1	Copies the drop source to the drop target.
fmDropEffectMove	2	Moves the drop source to the drop target.
fmDropEffectCopyOrMove	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.

UndoAction Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3mthUndoActionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3mthUndoActionX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3mthUndoActionA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3mthUndoActionS"}

Reverses the most recent action that supports the Undo command.

Syntax

Boolean = *object*.**UndoAction**

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

Part	Description
object	Required. A valid object.

Remarks

The Undo command in the user interface uses the **UndoAction** method. For example, if you paste text into a **TextBox**, you can use **UndoAction** to remove that text and restore the previous contents of the **TextBox**.

Not all user actions can be undone. If an action cannot be undone, the Undo command is unavailable following the action.

Note If the **CanUndo** property is **False**, the Undo command is not available in the user interface, and **UndoAction** is not valid in code.

If **UndoAction** is applied to a form, all changes to the current record are lost. If **UndoAction** is applied to a control, only the control itself is affected.

You must apply this method before the form or control is updated. You may want to include this method in a form's BeforeUpdate event or a control's Change event.

UndoAction is an alternative to using the SendKeys Statement to send the value of ESC in an event procedure.

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
object	Required. A valid object.
zPosition	Optional. A control's position, front or back, in the container's z-order.

Settings

The settings for *zPosition* are:

Constant	Value	Description
fmTop	0	Places the control at the front of the z-order. The control appears on top of other controls (default).
fmBottom	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 mode, 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
object	Required. A valid object.
String	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 property sheet 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
object	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":"f3proAlignmentS"}

Specifies the position of a control relative to its caption.

Syntax

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

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

Part	Description
object	Required. A valid object.
fmAlignment	Optional. Caption position.

Settings

The settings for *fmAlignment* are:

Constant	Value	Description
fmAlignmentLeft	0	Places the caption to the left of the control.
fmAlignmentRight	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
object	Required. A valid object.
Boolean	Optional. Whether the control is resized.

Settings

The settings for *Boolean* are:

Value	Description
True	Automatically resizes the control to display its entire contents.
False	Keeps the size of the control constant. Contents are clipped 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 ithe 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
object	Required. A valid object.
Boolean	Optional. Specifies whether an automatic tab occurs.

Settings

The settings for *Boolean* are:

Value	Description
True	Tab occurs.
False	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 a form 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
object	Required. A valid object.
Boolean	Optional. Specifies the basic unit used to extend a selection.

Settings

The settings for *Boolean* are:

Value	Description
True	Uses a word as the basic unit (default).
False	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
object	Required. A valid object.
Long	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 only see the background color of an object 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
object	Required. A valid object.
fmBackStyle	Optional. Specifies the control background.

Settings

The settings for *fmBackStyle* are:

Constant	Value	Description
fmBackStyleTransparent	0	The background is transparent.
fmBackStyleOpaque	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 a form. If **BackStyle** is **fmBackStyleTransparent**, you can see through the control and look at anything on the form located behind the control.

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

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

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

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
object	Required. A valid object name.
Boolean	Optional. Specifies the font style.
Currency	Optional. A number indicating the font size.
Integer	Optional. Specifies the font style.

The settings for *Boolean* are:

Value	Description
True	The text has the specified attribute (that is bold, italic, size, strikethrough or underline marks, or weight).
False	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 an object's border.

Syntax

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

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

Part	Description
object	Required. A valid object.
Long	Optional. A value or constant that determines the border 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 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 **SpecialEffect** property uses system colors exclusively to define its border colors. For Windows operating systems, system color settings are part of the **Control Panel** and are found in the **Appearance** tab of the **Display** folder. In Windows NT 3.51, system color settings are stored in the **Color** folder of the **Control Panel**.

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 or a form.

Syntax

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

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

Part	Description
object	Required. A valid object.
fmBorderStyle	Optional. Specifies the border style.

Settings

The settings for *fmBorderStyle* are:

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

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

Remarks

For a **Frame**, the **BorderStyle** property is ignored if the **SpecialEffect** property is *None*.

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
object	Required. A valid object.
Variant	Optional. Indicates how the **BoundColumn** value is selected.

Settings

The settings for *Variant* are:

Value	Description
0	Assigns the value of the **ListIndex** 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**.

If the control is bound to a data source, the value in the column specified by **BoundColumn** is stored in the data source named in the **ControlSource** property.

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).

BoundValue Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proBoundValueC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proBoundValueX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proBoundValueA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proBoundValueS"}

Contains the value of a control when that control receives the focus.

Syntax

object.**BoundValue** [= *Variant*]

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

Part	Description
object	Required. A valid object.
Variant	Optional. The current state or content of the control.

Settings

Control	Description
CheckBox	An integer value indicating whether the item is selected: Null Indicates the item is in a null state, neither selected nor cleared. –1 True. Indicates the item is selected. 0 False. Indicates the item is cleared.
OptionButton	Same as **CheckBox**.
ToggleButton	Same as **CheckBox**.
ScrollBar	An integer between the values specified for the **Max** and **Min** properties.
SpinButton	Same as **ScrollBar**.
ComboBox, ListBox	The value in the **BoundColumn** of the currently selected rows.
CommandButton	Always **False**.
MultiPage	An integer indicating the currently active page. Zero (0) indicates the first page. The maximum value is one less than the number of pages.
TextBox	The text in the edit region.

Remarks

BoundValue applies to the control that has the focus.

The contents of the **BoundValue** and **Value** properties are identical most of the time. When the user edits a control so that its value changes, the contents of **BoundValue** and **Value** are different until the change is final.

Several things occur when the user changes the value of a control. For example, if a user changes the text in a **TextBox**, the following things occur:

1. The **Change** event is initiated. At this time the **Value** property contains the new text and **BoundValue** contains the previous text.
2. The **BeforeUpdate** event is initiated.
3. The **AfterUpdate** event is initiated. The values for **BoundValue** and **Value** are once again identical, containing the new text.

BoundValue cannot be used with a multi-select list box.

Cancel Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proCancelC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proCancelX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proCancelS"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proCancelA"}

Returns or sets a value indicating whether a command button is the Cancel button on a form.

Syntax

object.**Cancel** [= *Boolean*]

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

Part	Description
object	Required. A valid object.
Boolean	Optional. Whether the object is the Cancel button.

Settings

The settings for *Boolean* are:

Value	Description
True	The **CommandButton** is the Cancel button.
False	The **CommandButton** is not the Cancel button (default).

Remarks

A **CommandButton** or an object that acts like a command button can be designated as the default command button. For OLE container controls, the **Cancel** property is provided only for those objects that specifically behave as command buttons.

Only one **CommandButton** on a form can be the Cancel button. Setting **Cancel** to **True** for one command button automatically sets it to **False** for all other objects on the form. When a **CommandButton's Cancel** property is set to **True** and the form is the active form, the user can choose the command button by clicking it, pressing ESC, or pressing ENTER when the button has the focus.

A typical use of **Cancel** is to give the user the option of canceling uncommitted changes and returning the form to its previous state.

You should consider making the Cancel button the default button for forms that support operations that can’t be undone (such as delete). To do this, set both **Cancel** and the **Default** property to **True**.

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**

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

Part	Description
object	Required. A valid object.

Return Values

The **CanPaste** property return values are:

Value	Description
True	The object underneath the mouse pointer can receive information pasted from the Clipboard (default).
False	The object underneath the mouse pointer cannot receive information pasted from the Clipboard.

Remarks

CanPaste is read-only.

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**.

CanRedo Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proCanRedoC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proCanRedoX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proCanRedoA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proCanRedoS"}

Indicates whether the most recent Undo can be reversed.

Syntax

object.**CanRedo**

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

Part	Description
object	Required. A valid object.

Return Values

The **CanRedo** property return values are:

Value	Description
True	The most recent Undo can be reversed.
False	The most recent Undo is irreversible.

Remarks

CanRedo is read-only.

To Redo an action means to reverse an Undo; it does not necessarily mean to repeat the last user action.

The following user actions illustrate using Undo and Redo:

1. Change the setting of an option button.
2. Enter text into a text box.
3. Click Undo. The text disappears from the text box.
4. Click Undo. The option button reverts to its previous setting.
5. Click Redo. The value of the option button changes.
6. Click Redo. The text reappears in the text box.

CanUndo Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proCanUndoC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proCanUndoX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proCanUndoA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proCanUndoS"}

Indicates whether the last user action can be undone.

Syntax

object.**CanUndo**

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

Part	Description
object	Required. A valid object.

Return Values

The **CanUndo** property return values are:

Value	Description
True	The most recent user action can be undone.
False	The most recent user action cannot be undone.

Remarks

CanUndo is read-only.

Many user actions can be undone with the Undo command. The **CanUndo** property indicates whether the most recent action can be undone.

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
object	Required. A valid object.
String	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 a form.

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 **Page** and **Tab** objects, it specifies the text that appears on the tab.

If a control's caption is too long, the caption is truncated. If a form'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
object	Required. A valid object.
Single	Optional. For **ClientHeight** and **ClientWidth**, specifies the height or width, in points, of the display area. For **ClientLeft** and **ClientTop**, specifies the distance, in points, from the top or left edge of the **TabStrip's** 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
object	Required. A valid object.
column	Optional. An integer with a range from 0 to one less than the total number of columns.
row	Optional. An integer with a range from 0 to one less than the total number of rows.
Variant	Optional. Specifies a single value, a column of values, or a two-dimensional array to load into a **ListBox** or **ComboBox**.

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. For example, you can set the **ControlSource** property of a text box to the value in the second column of a list 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
object	Required. A valid object.
Long	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. For an unbound data source, 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
object	Required. A valid object.
Boolean	Optional. Specifies whether the column headings are displayed.

Settings

The settings for *Boolean* are:

Value	Description
True	Display column headings.
False	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
object	Required. A valid object.
String	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.

ControlSource Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proControlSourceC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proControlSourceX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proControlSourceA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proControlSourceS"}

Identifies the data location used to set or store the **Value** property of a control.

Syntax

object.**ControlSource** [= *String*]

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

Part	Description
object	Required. A valid object.
String	Optional. Specifies the worksheet cell linked to the **Value** property of a control.

Remarks

The **ControlSource** property identifies a cell or field; it does not contain the data stored in the cell or field. If you change the **Value** of the control, the change is automatically reflected in the linked cell or field. Similarly, if you change the value of the linked cell or field, the change is automatically reflected in the **Value** of the control.

You cannot specify another control for the **ControlSource**. Doing so causes an error.

The default value for **ControlSource** is an empty string. If **ControlSource** contains a value other than an empty string, it identifies a linked cell or field. The contents of that cell or field are automatically copied to the **Value** property when the control is loaded.

Note If the **Value** property is **Null**, no value appears in the location identified by **ControlSource**.

ControlTipText Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proControlTipTextC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proControlTipTextX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proControlTipTextA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proControlTipTextS"}

Specifies text that appears when the user briefly holds the mouse pointer over a control without clicking.

Syntax

object.**ControlTipText** [= *String*]

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

Part	Description
object	Required. A valid object.
String	Optional. The text that appears when the user holds the mouse pointer over a control.

Remarks

The **ControlTipText** property lets you give users tips about a control in a running form. The property can be set during design time but only appears by the control during run time.

The default value of **ControlTipText** is an empty string. When the value of **ControlTipText** is set to an empty string, no tip is available for that control.

Count Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proCountC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proCountX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proCountS"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proCountA"}

Returns the number of objects in a collection.

Syntax

object.**Count**

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

Part	Description
object	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. For example, if a **MultiPage** contains two pages, the indexes of the pages are 0 and 1, and the value of **Count** is 2.

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
object	Required. A valid object.
Long	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**

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

Part	Description
object	Required. A valid object.

Return Values

The **CurTargetX** property retrieves 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 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

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
object	Required. A valid object.
Long	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.

Cycle Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proCycleC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proCycleX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proCycleS"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proCycleA"}

Specifies the action to take when the user leaves the last control on a **Frame** or **Page**.

Syntax

object.**Cycle** [= *fmCycle*]

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

Part	Description
object	Required. A valid object.
fmCycle	Optional. Specifies whether cycling includes controls nested in a **Frame** or **MultiPage**.

Settings

The settings for *fmCycle* are:

Constant	Value	Description
fmCycleAllForms	0	Cycles through the controls on the form and the controls of the **Frame** and **MultiPage** controls that are currently displayed on the form.
fmCycleCurrentForm	2	Cycles through the controls on the form, **Frame**, or **MultiPage**. The focus stays within the form, **Frame**, or **MultiPage** until the focus is explicitly set to a control outside the form, **Frame**, or **MultiPage**.

If you specify a non-integer value for **Cycle**, the value is rounded up to the nearest integer.

Remarks

The tab order identifies the order in which controls receive the focus as the user tabs through a form or subform. The **Cycle** property determines the action to take when a user tabs from the last control in the tab order.

The **fmCycleAllForms** setting transfers the focus to the the first control of the next **Frame** or **MultiPage** on the form when the user tabs from the last control in the tab order.

The **fmCycleCurrentForm** setting transfers the focus to the the first control of the same form, **Frame**, or **MultiPage** when the user tabs from the last control in the tab order.

Default Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proDefaultC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proDefaultX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proDefaultS"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proDefaultA"}

Designates the default command button on a form.

Syntax

object.**Default** [= *Boolean*]

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

Part	Description
object	Required. A valid object.
Boolean	Optional. Whether the command button is the default.

Settings

The settings for *Boolean* are:

Value	Description
True	The **CommandButton** is the default button.
False	The **CommandButton** is not the default button (default).

Remarks

A **CommandButton** or an object that acts like a command button can be designated as the default command button. Only one object on a form can be the default command button. Setting the **Default** property to **True** for one object automatically sets it to **False** for all other objects on the form.

To choose the default command button on an active form, the user can click the button, or press ENTER when no other **CommandButton** has the focus. Pressing ENTER when no other **CommandButton** has the focus also initiates the KeyUp event for the default command button.

Default is provided for OLE container controls that specifically act like **CommandButton** controls.

Tip You should consider making the Cancel button the default button for forms that support operations that can't be undone (such as delete). To do this, set both **Default** and the **Cancel** property to **True**.

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
object	Required. A valid object.
Long	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
object	Required. A valid object.
fmDragBehavior	Optional. Specifies whether the drag-and-drop feature is enabled.

Settings

The settings for *fmDragBehavior* are:

Constant	Value	Description
fmDragBehaviorDisabled	0	Does not allow a drag-and-drop action (default).
fmDragBehaviorEnabled	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.

DrawBuffer Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proDrawBufferC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proDrawBufferX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proDrawBufferA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proDrawBufferS"}

Specifies the number of pixels set aside for off-screen memory in rendering a frame.

Syntax

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

Part	Description
object	Required. A valid object name.
value	An integer from 16,000 through 1,048,576 equal to the maximum number of pixels the object can render off-screen.

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 object depends upon the screen resolution of the display. If you set a large value for **DrawBuffer**, performance will be slower. A large buffer only helps when several large images overlap.

Use the **Properties** window to specify the value of **DrawBuffer**.

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
object	Required. A valid object.
fmDropButtonStyle	Optional. The appearance of the drop button.

Settings

The settings for *fmDropButtonStyle* are:

Constant	Value	Description
fmDropButtonStylePlain	0	Displays a plain button, with no symbol.
fmDropButtonStyleArrow	1	Displays a down arrow (default).
fmDropButtonStyleEllipsis	2	Displays an ellipsis ().
fmDropButtonStyleReduce	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
object	Required. A valid object.
Boolean	Optional. Whether the object can respond to user-generated events.

Settings

The settings for *Boolean* are:

Value	Description
True	The control can receive the focus and respond to user-generated events, and is accessible through code (default).
False	The user cannot interact with the control by using the mouse, keystrokes, accelerators, or hotkeys. 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 form. 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 form. 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 form. 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 form. 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 **MultiPage** or **TabStrip**, the first page or tab in the control receives the focus. If the first page or tab of a **MultiPage** or **TabStrip** is disabled, the first enabled page or tab of that control receives the focus. If all pages or tabs of a **MultiPage** or **TabStrip** are disabled, the control is disabled and cannot receive the focus.

If a **Frame** is disabled, all controls it contains are disabled.

Clicking a disabled **ListBox** does not initiate the Click event.

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
object	Required. A valid object.
fmEnterFieldBehavior	Optional. The desired selection behavior.

Settings

The settings for *fmEnterFieldBehavior* are:

Constant	Value	Description
fmEnterFieldBehaviorSelectAll	0	Selects the entire contents of the edit region when entering the control (default).
fmEnterFieldBehaviorRecallSelection	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
object	Required. A valid object.
Boolean	Optional. Specifies the effect of pressing ENTER.

Settings

The settings for *Boolean* are:

Value	Description
True	Pressing ENTER creates a new line.
False	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
object	Required. A valid object.
Long	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 forms 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 **Frame**, **ForeColor** changes the color of the caption. 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** property syntax has these parts:

Part	Description
object	Required. A valid **OptionButton**.
String	Optional. The name of the group that includes the **OptionButton**. Use the same setting for all buttons in the group. The default setting is an empty string.

Remarks

To create a group of mutually exclusive **OptionButton** controls, you can put the buttons in a **Frame** on your form, or you can use the **GroupName** property. **GroupName** is more efficient for the following reasons:

- You do not have to include a **Frame** for each group. By not using a **Frame**, you reduce the number of controls on the form, and in turn, improve performance and reduce the size of the form.
- You have more design flexibility. If you use a **Frame** to create the group, all the buttons must be inside the **Frame**. If you want more than one group, you must have one **Frame** for each group. However, if you use **GroupName** to create the group, the group can include option buttons anywhere on the form. If you want more than one group, specify a unique name for each group; you can still place the individual controls anywhere on the form.
- You can create buttons with transparent backgrounds, which can improve the visual appearance of your form. The **Frame** is not a transparent control.

Regardless of which method you use to create the group of buttons, 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.

For example, assume your form includes some option buttons and a **MultiPage** that also includes option buttons. The option buttons on the **MultiPage** are one group and the buttons on the form are another group. The two groups do not affect each other. Changing the setting of a button on the **MultiPage** does not affect the buttons on the form.

Height, Width Properties

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proHeightC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proHeightX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proHeightS"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proHeightA"}

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
object	Required. A valid object.
Single	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 and the **OldHeight** or **OldWidth** property stores the previous 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 form.

If you move a control from one part of a form to another, the setting of **Height** or **Width** only changes 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 form 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.

HelpContextID Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proHelpContextIDC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proHelpContextIDX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proHelpContextIDA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proHelpContextIDS"}

Associates a specific topic in a custom Microsoft Windows Help file with a specific control.

Syntax

object.**HelpContextID** [= *Long*]

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

Part	Description
object	Required. A valid object.
Long	Optional. A positive integer specifies the context ID of a topic in the Help file associated with the object. Zero indicates no Help topic is associated with the object (default). Must be a valid context ID in the specified Help file.

Remarks

The topic identified by the **HelpContextID** property is available to users when a form is running. To display the topic, the user must either select the control or set focus to the control, and then press F1.

The **HelpContextID** property refers to a topic in a custom Help file you have created to describe your form or application. In Visual Basic, the custom Help file is a property of the project.

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
object	Required. A valid object.
Boolean	Optional. Whether the selected text remains highlighted even when the control does not have the focus.

Settings

The settings for *Boolean* are:

Value	Description
True	Selected text is not highlighted unless the control has the focus (default).
False	Selected text always appears highlighted.

Remarks

You can use the **HideSelection** property to maintain highlighted text when another form 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
object	Required. A valid object.
fmIMEMode	Optional. The mode of the Input Method Editor (IME).

Settings

The settings for *fmIMEMode* are:

Constant	Value	Description
fmIMEModeNoControl	0	Does not control IME (default).
fmIMEModeOn	1	IME on.
fmIMEModeOff	2	IME off. English mode.
fmIMEModeDisable	3	IME off. User can't turn on IME by keyboard.
fmIMEModeHiragana	4	IME on with Full-width Hiragana mode.
fmIMEModeKatakana	5	IME on with Full-width Katakana mode.
fmIMEModeKatakanaHalf	6	IME on with Half-width Katakana mode.
fmIMEModeAlphaFull	7	IME on with Full-width Alphanumeric mode.
fmIMEModeAlpha	8	IME on with Half-width Alphanumeric mode.
fmIMEModeHangulFull	9	IME on with Full-width Hangul mode.
fmIMEModeHangul	10	IME on with Half-width Hangul mode.

The **fmIMEModeNoControl** 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 toolbar of the IME. The other is with the **IMEMode** property of a control, 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 toolbar of the IME. Assume that you have designed a form 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 form, 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 anytime you click the control. This does not change the value of the property, it overrides the property until the next time you run the form.

Index Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proIndexC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proIndexX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proIndexS"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proIndexA"}

The position of a **Tab** object within a **Tabs** collection or a **Page** object in a **Pages** collection.

Syntax

object.**Index** [= *Integer*]

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

Part	Description
object	Required. A valid object.
Integer	Optional. The index of the currently selected **Tab** object.

Remarks

The **Index** property specifies the order in which tabs appear. Changing the value of **Index** visually changes the order of **Pages** in a **MultiPage** or **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 **MultiPage**, **Index** refers to a **Page** as well as the page's **Tab**. 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 a form. **InsideWidth** returns the width, in points, of the client region inside a form.

Syntax

object.**InsideHeight**
object.**InsideWidth**

The **InsideHeight** and **InsideWidth** property syntaxes have these parts:

Part	Description
object	Required. A valid object.

Remarks

The **InsideHeight** and **InsideWidth** properties are read-only. 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
object	Required. A valid object.
Boolean	Optional. Whether the list displays partial lines of text.

Settings

The settings for *Boolean* are:

Value	Description
True	The list resizes itself to display only complete items (default).
False	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.

TakeFocusOnClick Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proTakeFocusOnClickC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proTakeFocusOnClickX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proTakeFocusOnClickA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proTakeFocusOnClickS"}

Specifies whether a control takes the focus when clicked.

Syntax

object.**TakeFocusOnClick** [= *Boolean*]

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

Part	Description
object	Required. A valid object.
Boolean	Optional. Specifies whether a control takes the focus when clicked.

Settings

The settings for *Boolean* are:

Value	Description
True	The button takes the focus when clicked (default).
False	The button does not take the focus when clicked.

Remarks

The **TakeFocusOnClick** property defines only what happens when the user clicks a control. If the user tabs to the control, the control takes the focus regardless of the value of **TakeFocusOnClick**.

Use this property to complete actions that affect a control without requiring that control to give up focus. For example, assume your form includes a **TextBox** and a **CommandButton** that checks for correct spelling of text. You would like to be able to select text in the **TextBox**, then click the **CommandButton** and run the spelling checker without taking focus away from the **TextBox**. You can do this by setting the **TakeFocusOnClick** property of the **CommandButton** to **False**.

KeepScrollBarsVisible Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proKeepScrollBarsVisibleC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proKeepScrollBarsVisibleX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proKeepScrollBarsVisibleA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proKeepScrollBarsVisibleS"}

Specifies whether scroll bars remain visible when not required.

Syntax

object.**KeepScrollBarsVisible** [= *fmScrollBars*]

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

Part	Description
object	Required. A valid object.
fmScrollBars	Optional. Where scroll bars are displayed.

Settings

The settings for *fmScrollBars* are:

Constant	Value	Description
fmScrollBarsNone	0	Displays no scroll bars.
fmScrollBarsHorizontal	1	Displays a horizontal scroll bar.
fmScrollBarsVertical	2	Displays a vertical scroll bar.
fmScrollBarsBoth	3	Displays both a horizontal and a vertical scroll bar (default).

Remarks

If the visible region is large enough to display all the controls on an object such as a **Page** object or a form, scroll bars are not required. The **KeepScrollBarsVisible** property determines whether the scroll bars remain visible when they are not required.

If the scroll bars are visible when they are not required, they appear normal in size, and the scroll box fills the entire width or height of the scroll bar.

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
object	Required. A valid object.
Long	Optional. An integer that specifies the amount of change to the **Value** 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**.

LayoutEffect Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proLayoutEffectC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proLayoutEffectX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proLayoutEffectA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proLayoutEffectS"}

Specifies whether a control was moved during a layout change.

Syntax

object.**LayoutEffect**

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

Part	Description
object	Required. A valid object.

Return Values

The **LayoutEffect** property return values are:

Constant	Value	Description
fmLayoutEffectNone	0	The control was not moved.
fmLayoutEffectInitiate	1	The control moved.

Remarks

The **LayoutEffect** property is read-only and is available only in the Layout event. The Layout event is initiated by the **Move** method if the *Layout* argument is **True**.

The Layout event is not initiated when you change the settings of the **Left**, **Top**, **Height**, or **Width** properties of a control.

The Layout event sets **LayoutEffect** for any control that was involved in a move operation. For example, if you move a group of controls, **LayoutEffect** of each control is set.

Left, Top Properties

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proLeftC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proLeftX":1}
{ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proLeftA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proLeftS"}

The distance between a control and the left or top edge of the form that contains it.

Syntax

object.**Left** [= *Single*]
object.**Top** [= *Single*]

The **Left** and **Top** property syntaxes have these parts:

Part	Description
object	Required. A valid object.
Single	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 sytem 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 property sheet. When you print a form, 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**

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

Part	Description
object	Required. A valid object.

Remarks

The **LineCount** property is read-only.

Note A **ComboBox** will only have one line.

List Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proListC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proListX":1}
{ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proListA"} {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
object	Required. A valid object.
row	Required. An integer with a range from 0 to one less than the number of entries in the list.
column	Required. An integer with a range from 0 to one less than the number of columns.
Variant	Optional. The contents of the specified entry in the **ListBox** or **ComboBox**.

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** 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.

Use **List** to copy an entire two-dimensional array of values to a control. Use **AddItem** to load a one-dimensional array or to load an individual element.

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

object.**ListCount**

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

Part	Description
object	Required. A valid object.

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
object	Required. A valid object.
Variant	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.

The **ListIndex** value is also available by setting the **BoundColumn** property to 0 for a combo box or list box. If **BoundColumn** is 0, the underlying data source to which the combo box or list box is bound contains the same list index value as **ListIndex**.

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.

Syntax

object.**ListRows** [= *Long*]

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

Part	Description
object	Required. A valid object.
Long	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
object	Required. A valid object.
fmListStyle	Optional. The visual style of the list.

Settings

The settings for *fmListStyle* are:

Constant	Value	Description
fmListStylePlain	0	Looks like a regular list box, with the background of items highlighted.
fmListStyleOption	1	Shows option buttons, or check boxes for a multi-select 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 multi-select, 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
object	Required. A valid object.
Variant	Optional. The width of the list. A value of zero makes the list as wide as the **ComboBox**. 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
object	Required. A valid object.
Boolean	Optional. Whether the control can be edited.

Settings

The settings for *Boolean* are:

Value	Description
True	You can't edit the value.
False	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
object	Required. A valid object.
fmMatchEntry	Optional. The rule used to match entries in the list.

Settings

The settings for *fmMatchEntry* are:

Constant	Value	Description
fmMatchEntryFirstLetter	0	Basic matching. The control searches for the next entry that starts with the character entered. Repeatedly typing the same letter cycles through all entries beginning with that letter.
FmMatchEntryComplete	1	Extended matching. As each character is typed, the control searches for an entry matching all characters entered (default).
FmMatchEntryNone	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 Microsoft Forms, 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**

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

Part	Description
object	Required. A valid object.

Return Values

The **MatchFound** property return values are:

Value	Description
True	The contents of the **Value** property matches one of the records in the list.
False	The contents of **Value** 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
object	Required. A valid object.
Boolean	Optional. Whether the text entered must match an existing item in the list.

Settings

The settings for *Boolean* are:

Value	Description
True	The text entered must match an existing list entry.
False	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

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
object	Required. A valid object.
Long	Optional. A numeric expression specifying the maximum or minimum **Value** 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
object	Required. A valid object.
Long	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.

MouseIcon Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proMouseIconC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proMouseIconX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proMouseIconA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proMouseIconS"}

Assigns a custom icon to an object.

Syntax

object.**MouseIcon** = **LoadPicture(** *pathname* **)**

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

Part	Description
object	Required. A valid object.
pathname	Required. A string expression specifying the path and filename of the file containing the custom icon.

Remarks

The **MouseIcon** 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 **MouseIcon** property or load a picture from a file using the **LoadPicture** function.

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
object	Required. A valid object.
fmMousePointer	Optional. The shape you want for the mouse pointer.

Settings

The settings for *fmMousePointer* are:

Constant	Value	Description
fmMousePointerDefault	0	Standard pointer. The image is determined by the object (default).
fmMousePointerArrow	1	Arrow.
fmMousePointerCross	2	Cross-hair pointer.
fmMousePointerIBeam	3	I-beam.
fmMousePointerSizeNESW	6	Double arrow pointing northeast and southwest.
fmMousePointerSizeNS	7	Double arrow pointing north and south.
fmMousePointerSizeNWSE	8	Double arrow pointing northwest and southeast.
fmMousePointerSizeWE	9	Double arrow pointing west and east.
fmMousePointerUpArrow	10	Up arrow.
fmMousePointerHourglass	11	Hourglass.
fmMousePointerNoDrop	12	"Not" symbol (circle with a diagonal line) on top of the object being dragged. Indicates an invalid drop target.
fmMousePointerAppStarting	13	Arrow with an hourglass.
fmMousePointerHelp	14	Arrow with a question mark.
fmMousePointerSizeAll	15	Size all cursor (arrows pointing north, south, east, and west).
fmMousePointerCustom	99	Uses the icon specified by the **MouseIcon** property.

Remarks

Use the **MousePointer** property when you want to indicate changes in functionality as the mouse pointer passes over controls on a form. 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
object	Required. A valid object.
Boolean	Optional. Whether the control supports more than one line of text.

Settings

The settings for *Boolean* are:

Value	Description
True	The text is displayed across multiple lines (default).
False	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, including non-printing characters (such as carriage returns and new-lines).

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
object	Required. A valid object.
Boolean	Optional. Whether the control has more than one row of tabs.

Settings

The settings for *Boolean* are:

Value	Description
True	Allows more than one row of tabs.
False	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
object	Required. A valid object.
fmMultiSelect	Optional. The selection mode that the control uses.

Settings

The settings for *fmMultiSelect* are:

Constant	Value	Description
fmMultiSelectSingle	0	Only one item can be selected (default).
fmMultiSelectMulti	1	Pressing the SPACEBAR or clicking selects or deselects an item in the list.
fmMultiSelectExtended	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.

Name 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.**Name** [= *String*]
For all other controls and objects
object.**Name** [= *String*]

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

Part	Description
object	Required. A valid object.
String	Optional. The name you want to assign to the font or control.

Settings

Guidelines for assigning a string to **Name**, such as the maximum length of the name, vary from one application to another.

Remarks

For objects, the default value of **Name** consists of the object's class name followed by an integer. For example, the default name for the first **TextBox** you place on a form is TextBox1. The default name for the second **TextBox** is TextBox2.

You can set the **Name** property for a control from the control's property sheet or, for controls added at run time, by using program statements. If you add a control at design time, you cannot modify its **Name** property at run time.

Each control added to a form at design time must have a unique name.

For **Font** objects, **Name** identifies a particular typeface to use in the text portion of a control, object, or form. 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.

Object Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proObjectC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proObjectX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proObjectA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proObjectS"}

Overrides a standard property or method when a new control has a property or method of the same name.

Syntax

object.**Object**[.*property* |.*method*]

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

Part	Description
object	Required. The name of an object you have added to the Microsoft Forms Toolbox.
property	Optional. A property that has the same name as a standard Microsoft Forms property.
method	Optional. A method that has the same name as a standard Microsoft Forms method.

Remarks

Object is read-only.

If you add a new control to the Microsoft Forms Toolbox, it is possible that the added control will have a property or method with the same name as a standard Microsoft Forms property or method. The **Object** property lets you use the property or method from the added control, rather than the standard property or method.

OldHeight, OldWidth Properties

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proOldHeightC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proOldHeightX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proOldHeightA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proOldHeightS"}

Returns the previous height or width, in points, of the control.

Syntax

object.**OldHeight**
object.**OldWidth**

The **OldHeight** and **OldWidth** property syntaxes have these parts:

Part	Description
object	Required. A valid object.

Remarks

OldHeight and **OldWidth** are read-only.

The **OldHeight** and **OldWidth** properties are automatically updated when you move or size a control. If you change the size of a control, the **Height** and **Width** properties store the new height and **OldHeight** and **OldWidth** store the previous height.

These properties are valid only in the Layout event.

OldLeft, OldTop Properties

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proOldLeftC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proOldLeftX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proOldLeftA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proOldLeftS"}

Returns the distance, in points, between the previous position of a control and the left or top edge of the form that contains it.

Syntax

object.**OldLeft**
object.**OldTop**

The **OldLeft** and **OldTop** property syntaxes have these parts:

Part	Description
object	Required. A valid object.

Remarks

OldLeft and **OldTop** are read-only.

The **OldLeft** and **OldTop** properties are automatically updated when you move or size a control. If you move a control, the **Left** and **Top** properties store the new distance from the control to the left edge of its container and **OldLeft** and **OldTop** store the previous value of **Left**.

OldLeft and **OldTop** are valid only in the Layout event.

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
object	Required. A valid object.
fmOrientation	Optional. Orientation of the control.

Settings

The settings for *fmOrientation* are:

Constant	Value	Description
fmOrientationAuto	–1	Automatically determines the orientation based upon the dimensions of the control (default).
FmOrientationVertical	0	Control is rendered vertically.
FmOrientationHorizontal	1	Control is rendered horizontally.

Remarks

If you specify automatic orientation, the height and width of the control determine whether it appears horizontally or vertically. For example, if the control is wider than it is tall, it appears horizontally; if it is taller than it is wide, the control appears vertically.

Parent Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proParentC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proParentX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proParentA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proParentS"}

Returns the name of the form, object, or collection that contains a specific control, object, or collection.

Syntax

object.**Parent**

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

Part	Description
object	Required. A valid object.

Remarks

Parent is read-only.

Use the **Parent** property to access the properties, methods, or controls of an object's parent.

This property is useful in an application in which you pass objects as arguments. For example, you could pass a control variable to a general procedure in a module, and use **Parent** to access its parent form.

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
object	Required. A valid object.
String	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,"Specifics":"f3proPictureS"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proPictureA"}

Specifies the bitmap to display on an object.

Syntax

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

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

Part	Description
object	Required. A valid object.
pathname	Required. The full path to a picture file.

Remarks

While designing a form, you can use the control's property page to assign a bitmap to the **Picture** property. While running a form, 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 a control that supports opaque images. **Image** and **MultiPage** 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
object	Required. A valid object.
fmPictureAlignment	Optional. The position where the picture aligns with the control.

Settings

The settings for *fmPictureAlignment* are:

Constant	Value	Description
fmPictureAlignmentTopLeft	0	The top left corner.
fmPictureAlignmentTopRight	1	The top right corner.
fmPictureAlignmentCenter	2	The center.
fmPictureAlignmentBottomLeft	3	The bottom left corner.
fmPictureAlignmentBottomRight	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
object	Required. A valid object.
fmPicturePosition	Optional. How the picture aligns with its container.

Settings

The settings for *fmPicturePosition* are:

Constant	Value	Description
fmPicturePositionLeftTop	0	The picture appears to the left of the caption. The caption is aligned with the top of the picture.
fmPicturePositionLeftCenter	1	The picture appears to the left of the caption. The caption is centered relative to the picture.
fmPicturePositionLeftBottom	2	The picture appears to the left of the caption. The caption is aligned with the bottom of the picture.
fmPicturePositionRightTop	3	The picture appears to the right of the caption. The caption is aligned with the top of the picture.
fmPicturePositionRightCenter	4	The picture appears to the right of the caption. The caption is centered relative to the picture.
fmPicturePositionRightBottom	5	The picture appears to the right of the caption. The caption is aligned with the bottom of the picture.
fmPicturePositionAboveLeft	6	The picture appears above the caption. The caption is aligned with the left edge of the picture.
fmPicturePositionAboveCenter	7	The picture appears above the caption. The caption is centered below the picture (default).
fmPicturePositionAboveRight	8	The picture appears above the caption. The caption is aligned with the right edge of the picture.
fmPicturePositionBelowLeft	9	The picture appears below the caption. The caption is aligned

		with the left edge of the picture.
fmPicturePositionBelowCenter	10	The picture appears below the caption. The caption is centered above the picture.
fmPicturePositionBelowRight	11	The picture appears below the caption. The caption is aligned with the right edge of the picture.
fmPicturePositionCenter	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, form, or page.

Syntax

object.**PictureSizeMode** [= *fmPictureSizeMode*]

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

Part	Description
object	Required. A valid object.
fmPictureSizeMode	Optional. The action to take if the picture and the form or page that contains it are not the same size.

Settings

The settings for *fmPictureSizeMode* are:

Constant	Value	Description
fmPictureSizeModeClip	0	Crops any part of the picture that is larger than the form or page (default).
fmPictureSizeModeStretch	1	Stretches the picture to fill the form or page. This setting distorts the picture in either the horizontal or vertical direction.
fmPictureSizeModeZoom	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 form or page is smaller than the picture, this setting only shows the part of the picture that fits within the form or 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 a form or page.

Syntax

object.**PictureTiling** [= *Boolean*]

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

Part	Description
object	Required. A valid object.
Boolean	Optional. Whether a picture is repeated across a background.

Settings

The settings for *Boolean* are:

Value	Description
True	The picture is tiled across the background.
False	The picture is not tiled across the background (default).

Remarks

If a picture is smaller than the form or page that contains it, you can tile the picture on the form or page.

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 the form or page and down the height of the form or page. If **PictureSizeMode** is set to **fmPictureSizeModeClip**, the tiling pattern crops the last tile if it doesn't completely fit on the form or page.

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
object	Required. A valid object.
Boolean	Optional. Whether the scroll box is proportional or fixed.

Settings

The settings for *Boolean* are:

Value	Description
True	The scroll box is proportional in size to the scrolling region (default).
False	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.

RowSource Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proRowSourceC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proRowSourceX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proRowSourceA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proRowSourceS"}

Specifies the source providing a list for a **ComboBox** or **ListBox**.

Syntax

object.**RowSource** [= *String*]

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

Part	Description
object	Required. A valid object.
String	Optional. The source of the list for the **ComboBox** or **ListBox**.

Remarks

The **RowSource** property accepts worksheet ranges from Microsoft Excel.

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
object	Required. A valid object.
fmScrollBars	Optional. Where scroll bars should be displayed.

Settings

The settings for *fmScrollBars* are:

Constant	Value	Description
fmScrollBarsNone	0	Displays no scroll bars (default).
fmScrollBarsHorizontal	1	Displays a horizontal scroll bar.
fmScrollBarsVertical	2	Displays a vertical scroll bar.
fmScrollBarsBoth	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**.

ScrollHeight, ScrollWidth Properties

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proScrollHeightC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proScrollHeightX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proScrollHeightA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proScrollHeightS"}

Specify the height, in points, of the total area that can be viewed by moving the scroll bars on the control, form, or page.

Syntax

object.**ScrollHeight** [= *Single*]
object.**ScrollWidth** [= *Single*]

The **ScrollHeight** and **ScrollWidth** property syntaxes have these parts:

Part	Description
object	Required. A valid object.
Single	Optional. The height or width of the scrollable region.

ScrollLeft, ScrollTop Properties

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proScrollLeftC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proScrollLeftX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proScrollLeftA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proScrollLeftS"}

Specify the distance, in points, of the left or top edge of the visible form from the left or top edge of the logical form, page, or control.

Syntax

object.**ScrollLeft** [= *Single*]
object.**ScrollTop** [= *Single*]

The **ScrollLeft** and **ScrollTop** property syntaxes have these parts:

Part	Description
object	Required. A valid object.
Single	Optional. The distance from the edge of the form.

Remarks

The minimum value is zero; the maximum value is the difference between the value of the **ScrollWidth** property and the value of the **Width** property for the form or page.

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
object	Required. A valid object.
index	Required. An integer with a range from 0 to one less than the number of items in the list.
Boolean	Optional. Whether an item is selected.

Settings

The settings for *Boolean* are:

Value	Description
True	The item is selected.
False	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 multi-select 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 the currently selected **Tab** or **Page** object.

Syntax

object.**SelectedItem**

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

Part	Description
object	Required. A valid **TabStrip** or **MultiPage**.

Remarks

The **SelectedItem** property is read-only. Use **SelectedItem** to programmatically control the currently selected **Tab** or **Page** object. For example, you can use **SelectedItem** to assign values to properties of a **Tab** or **Page** 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
object	Required. A valid object.
Boolean	Optional. Whether clicking in the margin selects a line of text.

Settings

The settings for *Boolean* are:

Value	Description
True	Clicking in margin causes selection of text (default).
False	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 also prints.

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"}

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
object	Required. A valid object.
Long	Optional. A numeric expression specifying the number of characters selected. For **SelLength** and **SelStart**, the valid range of settings is 0 to the total number of characters in the edit area of a **ComboBox** or **TextBox**.

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
object	Required. A valid object.
Long	Optional. A numeric expression specifying the starting point of text selected. For **SelLength** and **SelStart**, the valid range of settings is 0 to the total number of characters in the edit area of a **ComboBox** or **TextBox**. 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
object	Required. A valid object.
String	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
object	Required. A valid object.
fmShowDropButtonWhen	Optional. The circumstances under which the drop-down button will be visible.

Settings

The settings for *fmShowDropButtonWhen* are:

Constant	Value	Description
fmShowDropButtonWhenNever	0	Do not show the drop-down button under any circumstances.
fmShowDropButtonWhenFocus	1	Show the drop-down button when the control has the focus.
fmShowDropButtonWhenAlways	2	Always show the drop-down button.

For a **ComboBox**, the default value is *fmShowDropButtonWhenAlways*; for a **TextBox**, the default value is *fmShowDropButtonWhenNever*.

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
object	Required. A valid object.
Long	Optional. An integer that specifies the amount of change to the **Value** 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"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proSpecialEffectS"}

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
object	Required. A valid object.
fmButtonEffect	Optional. The desired visual appearance for a **CheckBox**, **OptionButton**, or **ToggleButton**.
fmSpecialEffect	Optional. The desired visual appearance of an object other than a **CheckBox**, **OptionButton**, or **ToggleButton**.

Settings

The settings for *fmSpecialEffect* are:

Constant	Value	Description
fmSpecialEffectFlat	0	Object appears flat, distinguished from the surrounding form by a border, a change of color, or both. Default for **Image** and **Label**, valid for all controls.
fmSpecialEffectRaised	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.
fmSpecialEffectSunken	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 **CheckBox** and **OptionButton**, valid for all controls (default).
fmSpecialEffectEtched	3	Border appears to be carved around the edge of the control. Not valid for check boxes or option buttons.
fmSpecialEffectBump	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

For **ComboBox**, specifies how the user can choose or set the control's value. For **MultiPage** and **TabStrip**, identifies the style of the tabs on the control.

Syntax

For ComboBox
object.**Style** [= *fmStyle*]
For MultiPage and TabStrip
object.**Style** [= *fmTabStyle*]

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

Part	Description
object	Required. A valid object.
fmStyle	Optional. Specifies how a user sets the value of a **ComboBox**.
fmTabStyle	Optional. Specifies the tab style in a **MultiPage** or **TabStrip**.

Settings

The settings for *fmStyle* are:

Constant	Value	Description
fmStyleDropDownCombo	0	The **ComboBox** 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).
fmStyleDropDownList	2	The **ComboBox** behaves as a list box. The user must choose a value from the list.

The settings for *fmTabStyle* are:

Constant	Value	Description
fmTabStyleTabs	0	Displays tabs on the tab bar (default).
fmTabStyleButtons	1	Displays buttons on the tab bar.
fmTabStyleNone	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
object	Required. A valid object.
Single	Optional. The number of points of the height or width of the tabs on a **TabStrip** or **MultiPage**.

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 form's tab order.

Syntax

object.**TabIndex** [= *Integer*]

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

Part	Description
object	Required. A valid object.
Integer	Optional. An integer from 0 to one less than the number of controls on the form that have a **TabIndex** property. Assigning a **TabIndex** value of less than 0 generates an error. If you assign a **TabIndex** 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
object	Required. A valid object.
Boolean	Optional. The effect of pressing TAB.

Settings

The settings for *Boolean* are:

Value	Description
True	Pressing TAB inserts a tab character in the edit region.
False	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 **MultiPage** or **TabStrip**.

Syntax

object.**TabOrientation** [= *fmTabOrientation*]

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

Part	Description
object	Required. A valid object.
fmTabOrientation	Optional. Where the tabs will appear.

Settings

The settings for *fmTabOrientation* are:

Constant	Value	Description
fmTabOrientationTop	0	The tabs appear at the top of the control (default).
fmTabOrientationBottom	1	The tabs appear at the bottom of the control.
fmTabOrientationLeft	2	The tabs appear at the left side of the control.
fmTabOrientationRight	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
object	Required. A valid object.
Boolean	Optional. Whether the object is a tab stop.

Settings

The settings for *Boolean* are:

Value	Description
True	Designates the object as a tab stop (default).
False	Bypasses the object when the user is tabbing, although the object still holds its place in the actual tab order, as determined by the **TabIndex** property.

Tag Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proTagC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proTagX":1}
{ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proTagA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proTagS"}

Stores additional information about an object.

Syntax

object.**Tag** [= *String*]

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

Part	Description
object	Required. A valid object.
String	Optional. A string expression identifying the object. The default is a zero-length string ("").

Remarks

Use the **Tag** property to assign an identification string to an object without affecting other property settings or attributes.

For example, you can use **Tag** to check the identity of a form or control that is passed as a variable to a procedure.

Text Property

Returns or sets the text in a **TextBox**. Changes the selected row in a **ComboBox** or **ListBox**.

Syntax

object.**Text** [= *String*]

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

Part	Description
object	Required. A valid object.
String	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
object	Required. A valid object.
fmTextAlign	Optional. How text is aligned in the control.

Settings

The settings for *fmTextAlign* are:

Constant	Value	Description
fmTextAlignLeft	1	Aligns the first character of displayed text with the left edge of the control's display or edit area (default).
fmTextAlignCenter	2	Centers the text in the control's display or edit area.
fmTextAlignRight	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 only affects 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
object	Required. A valid object.
Variant	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**

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

Part	Description
object	Required. A valid object.

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
object	Required. A valid object.
Variant	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.

TransitionEffect Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proTransitionEffectC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proTransitionEffectX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proTransitionEffectA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proTransitionEffectS"}

Specifies the visual effect to use when changing from one page to another.

Syntax

object.**TransitionEffect** [= *fmTransitionEffect*]

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

Part	Description
object	Required. A valid object.
fmTransitionEffect	Optional. The transition effect you want between pages.

Settings

The settings for *fmTransitionEffect* are:

Constant	Value	Description
fmTransitionEffectNone	0	No special effect (default).
fmTransitionEffectCoverUp	1	The new page covers the old page, moving from the bottom to the top.
fmTransitionEffectCoverRightUp	2	The new page covers the old page, moving from the bottom left corner to the top right corner.
fmTransitionEffectCoverRight	3	The new page covers the old page, moving from the left edge to the right.
fmTransitionEffectCoverRightDown	4	The new page covers the old page, moving from the top left corner to the bottom right corner.
fmTransitionEffectCoverDown	5	The new page covers the old page, moving from the top to the bottom.
fmTransitionEffectCoverLeftDown	6	The new page covers the old page, moving from the top right corner to the bottom left corner.
fmTransitionEffectCoverLeft	7	The new page covers the old page, moving from the right to the left.
fmTransitionEffectCoverLeftUp	8	The new page covers the old page, moving from the bottom right corner to the top left corner.
fmTransitionEffectPushUp	9	The new page pushes the old page out of view, moving

		from the bottom to the top.
fmTransitionEffectPushRight	10	The new page pushes the old page out of view, moving from the left to the right.
fmTransitionEffectPushDown	11	The new page pushes the old page out of view, moving from the top to the bottom.
fmTransitionEffectPushLeft	12	The new page pushes the old page out of view, moving from the right to the left.

Remarks

Use the **TransitionPeriod** property to specify the duration of a transition effect.

TransitionPeriod Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proTransitionPeriodC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proTransitionPeriodX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proTransitionPeriodA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proTransitionPeriodS"}

Specifies the duration, in milliseconds, of a transition effect.

Syntax

object.**TransitionPeriod** [= *Long*]

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

Part	Description
object	Required. A valid object.
Long	Optional. How long it takes to complete the transition from one page to another.

Remarks

Any integer from zero to 10000 is a valid setting for this property. Setting the **TransitionPeriod** property to zero disables the transition effect; setting **TransitionPeriod** to 10000 creates a 10-second transition.

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** or **ToggleButton**.

Syntax

object.**TripleState** [= *Boolean*]

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

Part	Description
object	Required. A valid object.
Boolean	Optional. Whether the control supports the Null state.

Settings

The settings for *Boolean* are:

Value	Description
True	The button clicks through three states.
False	The button only supports True and False (default).

Remarks

Although the **TripleState** property exists on the **OptionButton**, the property is disabled. Regardless of the value of **TripleState**, you cannot set the control to **Null** through the user interface.

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**.

A control set to **Null** does not initiate the Click event.

Regardless of the property setting, the **Null** value can always be assigned programmatically to a **CheckBox** or **ToggleButton**, causing that control to appear shaded.

Value Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proValueC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proValueX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proValueS"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proValueA"}

Specifies the state or content of a given control.

Syntax

object.**Value** [= *Variant*]

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

Part	Description
object	Required. A valid object.
Variant	Optional. The state or content of the control.

Settings

Control	Description
CheckBox	An integer value indicating whether the item is selected: Null Indicates the item is in a null state, neither selected nor cleared. –1 True. Indicates the item is selected. 0 False. Indicates the item is cleared.
OptionButton	Same as **CheckBox**.
ToggleButton	Same as **CheckBox**.
ScrollBar	An integer between the values specified for the **Max** and **Min** properties.
SpinButton	Same as **ScrollBar**.
ComboBox, ListBox	The value in the **BoundColumn** of the currently selected rows.
CommandButton	Always **False**.
MultiPage	An integer indicating the currently active page. Zero (0) indicates the first page. The maximum value is one less than the number of pages.
TextBox	The text in the edit region.

Remarks

For a **CommandButton**, setting the **Value** property to **True** in a macro or 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 multi-select list box.

VerticalScrollbarSide Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proVerticalScrollbarSideC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proVerticalScrollbarSideX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proVerticalScrollbarSideA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proVerticalScrollbarSideS"}

Specifies whether a vertical scroll bar appears on the right or left side of a form or page.

Syntax

object.**VerticalScrollbarSide** [= *fmVerticalScrollbarSide*]

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

Part	Description
object	Required. A valid object.
fmVerticalScrollbarSide	Optional. Where the scroll bar should appear.

Settings

The settings for *fmVerticalScrollbarSide* are:

Constant	Value	Description
fmVerticalScrollbarSideRight	0	Puts the scroll bar on the right side (default).
fmVerticalScrollBarSideLeft	1	Puts the scroll bar on the left side.

Remarks

The **VerticalScrollBarSide** property is is particularly useful if the form will be used in an environment where reading occurs from right to left.

Visible Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proVisibleC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proVisibleX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proVisibleS"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proVisibleA"}

Specifies whether an object is visible or hidden.

Syntax

object.**Visible** [= *Boolean*]

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

Part	Description
object	Required. A valid object.
Boolean	Optional. Whether the object is visible or hidden.

Settings

The settings for *Boolean* are:

Value	Description
True	Object is visible (default).
False	Object is hidden.

Remarks

Use the **Visible** property to control access to information without displaying it. For example, you could use the value of a control on a hidden form as the criteria for a query.

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
object	Required. A valid object.
Boolean	Optional. Whether the control expands to fit the text.

Settings

The settings for *Boolean* are:

Value	Description
True	The text wraps (default).
False	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**.

Zoom Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3proZoomC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"f3proZoomX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3proZoomA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3proZoomS"}

Specifies how much to change the size of a displayed object.

Syntax

object.**Zoom** [= *Integer*]

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

Part	Description
object	Required. A valid object.
Integer	Optional. The percentage to increase or decrease the displayed image.

Remarks

The value of the **Zoom** property specifies a percentage of image enlargement or reduction by which an image display should change. Values from 10 to 400 are valid. The value specified is a percentage of the object's original size; thus, a setting of 400 means you want to enlarge the image to four times its original size (or 400 percent), while a setting of 10 means you want to reduce the image to one-tenth of its original size (or 10 percent).

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.

bound

Describes a control whose contents are associated with a particular data source, such as a cell or cell range in a worksheet.

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.

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. Microsoft Forms 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.

data source

The location of data to which a control is bound, for example, a cell in a worksheet. The current value of the data source can be stored in the **Value** property of a control. However, the control does not store the data; it only displays the information that is stored in the data source.

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.

enumerated constant

You can find additional information for an enumerated data item in the description of the property, method, or event that uses the enumeration.

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 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 property sheet.

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.

system colors

Colors that are 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.

unbound

Describes a control that is not related to a worksheet cell. In contrast, a bound control is a data source for a worksheet cell that provides access to display and edit the value of a control.

What is a MultiPage?

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conWhatIsMultiPageC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3conWhatIsMultiPageS"}

A **MultiPage** is a control that contains a collection of one or more pages.

Each **Page** of a **MultiPage** is a form that contains its own controls, and as such, can have a unique layout. Typically, the pages in a **MultiPage** have tabs so the user can select the individual pages.

By default, a **MultiPage** includes two pages, called Page1 and Page2. Each of these is a **Page** object, and together they represent the **Pages** collection of the **MultiPage**. If you add more pages, they become part of the same **Pages** collection.

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.

Should I use a MultiPage or a TabStrip?

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conShouldIUseMultiPageOrTabStripC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3conShouldIUseMultiPageOrTabStripS"}

If you use a single layout for data, use a **TabStrip** and map each set of data to its own **Tab**. If you need several layouts for data, use a **MultiPage** and assign each layout to its own **Page**.

Unlike a **Page** of a **MultiPage**, the client region of a **TabStrip** is not a separate form, but a portion of the form that contains the **TabStrip**. The border of a **TabStrip** defines a region of the form that you can associate with the tabs. When you place a control in the client region of a **TabStrip**, you are adding a control to the form that contains the **TabStrip**.

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"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howCreateStandardListBoxS"}

1. In the **Properties** window, select the **ListStyle** property.
2. Click the drop-down arrow to display a list of available styles.
3. From the list, choose **Plain**.

Create a list box with option buttons or check boxes

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howCreateListBoxWithOptionButtonsOrCheckBoxesC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howCreateListBoxWithOptionButtonsOrCheckBoxesS"}

1. In the **Properties** window, select the **ListStyle** property.
2. Click the drop-down arrow to display a list of available styles.
3. 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 a form, **Frame**, or **Page**.

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 form 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 forms.

Add a customized control to the Toolbox

{ewc HLP95EN.DLL,DYNALINK,"See Also":" f3howAddCustomizedControlToToolboxC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":" f3howAddCustomizedControlToToolboxS"}

1 Place a control on your form and customize it.

For example, to create an **OK** button, place a **CommandButton** on the form, set its **Caption** property to **OK** and set its **Default** property to **True**.

3 Select the customized control.

4 Drag the control to the Toolbox.

Note When you drag a control onto the Toolbox, you only transfer property values. Any code you have written for that control does not transfer with the control. You must write new code for the icon or copy code from the control on the form to the control on the Toolbox.

Add a control to a form

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conAddControlToFormC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3conAddControlToFormS"}

Use any of the following methods to add a control from the Toolbox to your form. You can also use these methods to insert a control in a **Frame**, **TabStrip**, or **MultiPage** on the form.

- Click a control in the Toolbox and then click in the form. The control appears in its default size. You can then drag the control to change its size.
- Drag a control from the Toolbox to the form. The control appears in its default size.
- Double-click the control in the Toolbox, and then click in the form once for each control you want to create. For example, to create four command buttons, double-click the **CommandButton** in the Toolbox and then click four times in the form.

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.

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 bitmap, choose **Load Picture**. Then identify the file that contains the bitmap you want to use as the icon. If you attempt to load a picture that is larger than an icon, an error occurs.

What is a ToolTip?

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conWhatIsToolTipC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3conWhatIsToolTipS"}

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 a form 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.

Note The ToolTip is information provided by Microsoft Forms to forms developers in design mode. Each control has a property, **ControlTipText**, that allows forms developers to give a “ToolTip” to end users while the application is running.

Customize a ToolTip in the Toolbox

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howCustomizeToolTipInToolboxC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howCustomizeToolTipInToolboxS"}

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"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howSetToolTipForPageOfToolboxS"}

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"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howChangeOrderOfToolboxPagesS"}

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.
1. Click OK.

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. The pointer changes to a double-ended arrow.
2. When the double-ended arrow appears, drag the Toolbox to change its size.

Assign a custom Help topic to a control

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howAssignCustomHelpTopicToControlC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howAssignCustomHelpTopicToControlS"}

This procedure assumes you have created a custom Help file and associated it with your project. The procedure for associating a Help file with a project depends on your project environment.

1. Select a control for which you have written a custom Help topic.
2. In the **Properties** window, select the **HelpContextID** property.
3. Enter the context ID of the appropriate topic from your custom Help file.

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"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howAssignAcceleratorKeyS"}

1 In design mode, select the control on the form.

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 Page or Tab

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howAssignAcceleratorKeyForPageOrTabC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howAssignAcceleratorKeyForPageOrTabS"}

1. In design mode, select an individual **Page** or **Tab**. Be sure to select the **Page** or **Tab**, not the associated **MultiPage** or **TabStrip**. When a **Page** or **Tab** is selected, a rectangle appears around the caption of the **Page** or **Tab**.
2. Right-click the selected **Page** or **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 control tip to a Page or Tab

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howAssignControlTipToPageOrTabC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howAssignControlTipToPageOrTabS"}

1. Select an individual **Page** or **Tab**.
   Be sure to select an individual **Page** or **Tab**, not the corresponding **MultiPage** or **TabStrip**. When a **Page** or **Tab** is selected, a rectangle appears around its caption.
2. Right-click the selected **Page** or **Tab**.
3. From the shortcut menu, choose **Rename**.
4. In the **ControlTipText** field, type the string you want to use as the control tip.
5. Click OK.

Tip To assign a control tip for a **MultiPage** or **TabStrip**, use the **ControlTipText** property.

Assign a control tip to a control

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howAssignControlTipToControlC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howAssignControlTipToControlS"}

1. Select the control.
2. In the **Properties** window, select the **ControlTipText** property.

   You can also set the value of **ControlTipText** through code.
3. Enter the string you want to use as the control tip.

Tip To assign a control tip for a **Page** or **Tab**, use the **Rename** command on the shortcut menu of the **Page** or **Tab**.

What is a control tip?

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conWhatIsControlTipC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3conWhatIsControlTipS"}

A control tip is 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 very similar to a ToolTip. The difference is that Microsoft Forms provides ToolTips to developers at design time, and developers provide control tips to end-users at run time.

If you assign a control tip to a **MultiPage** or a **TabStrip**, control tips for the individual **Page** or **Tab** objects within the **MultiPage** or **TabStrip** are not displayed.

Assign a caption

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howAssignCaptionC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howAssignCaptionS"}

To assign a caption to a **CheckBox**, **CommandButton**, **Frame**, **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 **Page** or **Tab**:

1. Select the **MultiPage** or **TabStrip** that contains the **Page** or **Tab**.
2. Select the individual **Page** or **Tab**. When the **Page** or **Tab** is selected, a rectangle appears around its caption.
3. Right-click the selected **MultiPage** or **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**, **Frame**, **Label**, **OptionButton**, and **ToggleButton**. The **Page** and **Tab** objects that are part of the **MultiPage** and **TabStrip** also can have captions.

Set the tab order using the Tab Order dialog box

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howSetTabOrderUsingDialogBoxC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howSetTabOrderUsingDialogBoxS"}

1. Make sure no controls are selected.
2. Right-click in the form, but not on a control.
3. From the shortcut menu, choose **Tab Order**.
4. Select the name of a control you want to reposition in the tab order.
5. Choose **Move Up** or **Move Down** until the control name is in the appropriate position in the tab order.

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 form.

The tab index of the first control in the tab order is 0; the tab index of the second is 1, and so on.

3 Select a control in the tab order.

4 In the **Properties** window, select the **TabIndex** property.

5 Enter the appropriate number to identify the control's position in the tab order.

Change the order of pages in a MultiPage or TabStrip

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howChangeOrderOfPagesInMultiPageOrTabStripC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howChangeOrderOfPagesInMultiPageOrTabStripS"}

1. Select any page in the **MultiPage** or **TabStrip**.
2. Right-click the caption of the page.
3. From the shortcut menu, choose **Move**.
4. In the **Move** dialog box, select the **Page** 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 form

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howChangeSizeOfFormC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howChangeSizeOfFormS"}

At design time:

- Drag the sizing handle of the form until the form is the size you want.

At run time:

- Set the form's **Height** and **Width** properties to the appropriate values.

Change the location of the form

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howChangeLocationOfFormC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howChangeLocationOfFormS"}

Through the user interface:

- Drag the title bar until the form is where you want it.

At design time:

- Set the form's **Left** and **Top** properties to the appropriate values. You can set these properties through the **Properties** window or through code.

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 Microsoft Forms:

- 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 does 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"}

Microsoft Forms 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 form. 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"}

1. From the **Tools** menu, choose **Options**.
2. Select the **General** tab sheet.
3. Do one of the following:
   - To show the grid, check the **Show Grid** box.
   - To hide the grid, clear the **Show Grid** box.
1. Click OK.

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**.

Microsoft Forms adjusts the size of the selected control so that each corner aligns 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**.

Microsoft Forms sets the size of the control 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, choose **Make Same Size**.
4. From the cascading menu, choose 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, choose **Align**.
4. From the cascading menu, choose 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, choose **Horizontal Spacing** or **Vertical Spacing**.
3. From the cascading menu, choose 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.

Arrange buttons

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howArrangeButtonsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howArrangeButtonsS"}

1. Select the **CommandButton** controls to arrange.
2. From the **Format** menu, choose **Arrange Buttons**.
3. From the cascading menu, choose one of the following:
   - **Bottom**—to put the buttons in a row starting at the bottom left corner of the form and align the bottoms of all buttons.
   - **Right**—to put the buttons in a column starting at the upper right corner of the form and align the right edges of all buttons.

After you arrange the buttons, use either **Horizontal Spacing** or **Vertical Spacing** on the **Format** menu to adjust the spacing between the buttons.

Tip Select a small grid size before choosing this command to position the buttons close to the bottom or right of the form. Changing the grid size after the buttons are in place will not change their position.

Center controls in a form

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howCenterInFormC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howCenterInFormS"}

1. Select the controls or groups to center.
2. From the **Format** menu, choose **Center in Form**.
3. From the cascading menu, choose one of the following:
   - **Horizontally**
   - **Vertically**

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 a form that you treat as a single unit. You can include any control on the form in a group. Once controls belong to a group, you can work with the entire group, or you can select a single object.

Microsoft Forms 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 **Pages (Page)**.
- Break up the group so each control is independent of the others. For more information, click **Pages (Page)**.
- Display the group's shortcut menu, which provides quick access to commands that affect the group. For more information, click **Pages (Page)**.
- 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 **Pages (Page)**.

Transparency in Microsoft Forms

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conTransparencyInMSFormsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3conTransparencyInMSFormsS"}

Microsoft Forms supports transparency in two areas: the background of certain controls, and in bitmaps 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 form. This is useful if you have a decorative background on the form 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 **Pages (Page)**.

You can display a bitmap on many controls in Microsoft Forms. Certain controls support transparent bitmaps, that is, bitmaps in which one or more background color is transparent. Bitmap transparency is not controlled by any control property; it is controlled by the color of the lower-left pixel in the image. Microsoft Forms does not provide a way to edit a bitmap and make it transparent; you must use a picture editor for this purpose.

In Microsoft Forms, bitmaps 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 a control that supports opaque images.

If you use a transparent bitmap on a control that does not support transparent bitmaps, the bitmap will display correctly but you won't be able to see what's behind it. In Microsoft Forms, the following controls do not support transparent bitmaps:

- The form window (**UserForm**)
- **Frame**
- **Image**
- **MultiPage**

What is a shortcut menu?

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conWhatIsShortcutMenuC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3conWhatIsShortcutMenuS"}

A shortcut menu is a menu that appears when you right-click an object. In Microsoft Forms, 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 a form.
- Groups of controls (groups created with the **Group** command).
- Containers (such as UserForm).
- Individual **Page** objects in a **MultiPage**.
- 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 **MultiPage** or a **Page**, click Pages (Page).

For more information on displaying the shortcut menu for a **TabStrip** or a **Tab**, click Pages (Page).

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.

Similarly, you can control which column of values is used for the control when the user makes a selection by specifying the column number in the **BoundColumn** 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.

Show or hide ToolTips

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howShowHideToolTipsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howShowHideToolTipsS"}

1. From the **Tools** menu, choose **Options**.
2. Select the **General** tab sheet.
3. Do one of the following:
   - To display ToolTips, check the **Show ToolTips** box.
   - To hide ToolTips, clear the **Show ToolTips** box.

Object model for Microsoft Forms

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conObjectModelMSFormsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3conObjectModelMSFormsS"}

The Microsoft Forms object model includes the following types of object:

- Controls
- Collections
- Objects (within collections)

Each element of the Microsoft Forms 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.

Microsoft Forms has three collections:

Controls collection—contains all the controls on a form, **Frame**, or **Page**.

Pages collection—contains all the **Page** objects in a **MultiPage**. Each **MultiPage** has its own distinct **Pages** collection.

Tabs collection—contains all the **Tab** objects in a **TabStrip**. Each **TabStrip** has its own distinct **Tabs** collection.

Ways to create an option group

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conWaysToCreateOptionGroupC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3conWaysToCreateOptionGroupS"}

By default, all **OptionButton** controls on a container (such as a form, a **MultiPage**, or a **Frame**) are part of a single option group. This means that selecting one of the buttons automatically sets all other option buttons on the form to **False**.

If you want more than one option group on the form, there are two ways to create additional groups:

- Use the **GroupName** property to identify related buttons.
- Put related buttons in a **Frame** on the form.

The first method is recommended over the second because it reduces the number of controls required in the application. This reduces the disk space required for your application and can improve the performance of your application as well.

Note A **TabStrip** is not a container. Option buttons in the **TabStrip** are included in the form'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 form. Note that option buttons in a **MultiPage** or **Frame** will automatically form an option group.
2. Identify the buttons for each group you want to create.
3. Enter a value for the **Name** 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"}

Microsoft Forms provides 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 the controls you want to reposition.
2. From the **Format** menu, choose **Order**.
3. From the cascading menu, select **Bring to Front** or **Send to Back**.

To adjust a control one position in the z-order:

1. Select the controls you want to reposition.
2. From the **Format** menu, choose **Order**.
3. From the cascading menu, select **Bring Forward** or **Send 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 **Send Backward** on the shortcut menu, you won't be able to Undo or Redo that action.

The **Bring to Front**, **Bring Forward**, **Send to Back**, and **Send Backward** menu choices let you change the z-order of a control relative to other controls. If the form includes any **ListBox**, **Frame**, or **MultiPage** controls, those controls automatically move as close as possible to the top of the stack. For example, applying **Send Backward** to a **ListBox**, **Frame**, or **MultiPage** moves the control below other **ListBox**, **Frame**, or **MultiPage** controls, but will not move it below any other type of control in the stack. Similarly, applying **Bring Forward** to a control other than a **ListBox**, **Frame**, or **MultiPage** will move the control closer to to top of the stack, but will not move it above any **ListBox**, **Frame**, or **MultiPage** in the stack.

Visually, this means that if a **ListBox**, **Frame**, or **MultiPage** and any other Microsoft Forms control are in the same location on a form, the **ListBox**, **Frame**, or **MultiPage** will always appear on top of the other control. If a **ListBox**, **Frame**, or **MultiPage** is in the same place as another **ListBox**, **Frame**, or **MultiPage**, the z-order of the controls determines which control appears on top of the other.

Create a transparent control

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howCreateTransparentControlC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howCreateTransparentControlS"}

1. Put the basic control onto the form.
2. View the control's properties.
3. Set the **BackStyle** property to **Transparent**.
4. If the control supports the **BorderStyle** property, set it to **None**.

Note When you make a control transparent, the background color does not display, so the **BackColor** property is ignored. However, the setting for **BackColor** is not changed when a control is transparent.

Delete a bitmap from a control

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howDeleteBitmapFromControlC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howDeleteBitmapFromControlS"}

In the **Properties** window:

1. Highlight the value of the **Picture** property (the word “bitmap”).
2. Press DELETE.

Or, in code:

- Enter the following statement: Object.Picture = LoadPicture(“”)

Assign a bitmap to a control

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howAssignBitmapToControlC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howAssignBitmapToControlS"}

In the **Properties** window:

1. Choose the **Picture** property.
2. In the **Picture** dialog box, enter the name of the picture and its location.

If the picture is larger than the control, Microsoft Forms scales the picture 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** property assigns a bitmap 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**.
2. In the **Options** dialog box, choose the **General** page.
3. In the **Form Grid Settings** group, specify the size you want for each grid block. Specifying smaller numbers results in smaller grid blocks.

Tip If you use the **Arrange Buttons** command to position command buttons in your application, try a small grid setting. This will allow you to position the buttons closer to the edge of the form.

Create a control group

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howCreateControlGroupC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howCreateControlGroupS"}

1. In the form, select each control you want to include in the group.
2. From the **Format** 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.

3 Click one of the sizing handles and drag it to change the size of the rectangle.

4 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"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howBreakUpControlGroupS"}

1. Select the group.
2. From the **Format** menu, choose **Ungroup**.

Display a group’s shortcut menu

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howDisplayGroupsShortcutMenuC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howDisplayGroupsShortcutMenuS"}

1 Select the group.

2 Right-click anywhere inside the rectangle that surrounds the group.

Tip Click anywhere in the group, but not on the shortcut menu, to make the shortcut menu go away if you don’t want to use any of the commands on the menu.

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 a MultiPage or Page

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howDisplayShortcutMenuForMultiPageOrPageC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howDisplayShortcutMenuForMultiPageOrPageS"}

1. Make sure the form isn't running.
2. Do one of the following:
   - To display the shortcut menu of an individual **Page**, right-click the caption of the appropriate page.
   - To display the shortcut menu of the entire **MultiPage**, right-click anywhere in the control, but not on the caption of any **Page** in the control.

Display the shortcut menu for a TabStrip or Tab

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3howDisplayShortcutMenuForTabStripOrTabC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3howDisplayShortcutMenuForTabStripOrTabS"}

1. Make sure the form isn't running.
2. Do one of the following:
   - To display the shortcut menu of an individual **Tab**, select the appropriate tab.

   When the tab is selected, a dotted rectangle appears around its caption.

   Right-click the selected caption.
   - To display the shortcut menu of the **TabStrip**, right-click anywhere in the control, but not on the caption of any **Tab** in the control.

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 both the selected state and 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"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3conTipsOnSelectingMultipleControlsS"}

You can select more than one control in three ways:

- SHIFT+CLICK: Microsoft Forms creates an invisible selection rectangle around the selected controls and puts sizing handles on all controls within that rectangle.
- CTRL+CLICK: sizing handles only appear on the selected controls, not on controls within the surrounding rectangle. Occasionally, this method may select additional controls that are near to or adjacent to the selected controls. In that case, 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.

When you select multiple controls, one of the selected controls becomes a reference for the rest of the selected controls and is called the dominant control.

Tips on setting the dominant control

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conTipsOnSettingDominantControlC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3conTipsOnSettingDominantControlS"}

You can set the dominant control in one of the following ways when selecting multiple controls:

- SHIFT+CLICK: The dominant control is the first control you select using SHIFT+CLICK.
- CTRL+CLICK: The dominant control is the last control you select using CTRL+CLICK.
- **Select Objects** pointer on the Toolbox: The dominant control is nearest the mouse pointer when you begin drawing the rectangle over the controls you want to select.

If you CTRL+CLICK twice on a selected control, that control becomes the dominant control.

Undo and Redo in Microsoft Forms

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conUndoRedoInMicrosoftFormsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3conUndoRedoInMicrosoftFormsS"}

Microsoft Forms supports multiple levels of **Undo** and **Redo** commands. This means you can undo a series of actions, not just a single action.

CTRL+Z is the shortcut key for **Undo**; CTRL+Y is the shortcut key for **Redo**.

You cannot 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 will not be able to undo or redo that action.

ByVal References in Microsoft Forms

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3conByValReferencesInMicrosoftFormsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3conByValReferencesInMicrosoftFormsS"}

The ByVal keyword in Microsoft Forms indicates that an argument is passed as a value; this is the standard meaning of ByVal in Visual Basic. However, in Microsoft Forms, you can use ByVal with a ReturnBoolean, ReturnEffect, ReturnInteger, or ReturnString object. When you do, the value passed is not a simple data type; it is a pointer to the object.

When used with these objects, ByVal refers to the object, not the method of passing parameters. Each of the objects listed above has a **Value** property that you can set. You can also pass that value into and out of a function. Because you can change the values of the object's members, events produce results consistent with ByRef behavior, even though the event syntax says the parameter is ByVal.

Assigning a value to an argument associated with a ReturnBoolean, ReturnEffect, ReturnInteger, or ReturnString is no different from setting the value of any other argument. For example, if the event syntax indicates a *Cancel* argument used with the ReturnBoolean object, the statement `Cancel=True` is still valid, just as it is with other data types.

The Rename dialog box

Contains the **Accelerator**, **Caption**, and **ControlTipText** property settings for the individual page or tab that has the focus. Contains the **Caption** and **ToolTipText** property settings 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 **Page** or **Tab**. The caption is the text in the tab area of a **Page** or **Tab**, or the current Toolbox page. The **ControlTipText** is a brief description of a control that appears when the user holds the mouse pointer over the control without clicking. The **ToolTipText** is a brief description of a control that appears when the user holds the mouse pointer over the current Toolbox page without clicking.

To set an accelerator for the **Page** or **Tab**:

- Enter a single character for **Accelerator**.

To rename the **Page** or **Tab**:

- Enter a new value for **Caption**.

To define ControlTipText for the **Page** or **Tab**:

- Enter a new value for **ControlTipText**.

To define ToolTipText for the current Toolbox page:

- Enter a new value for **ToolTipText**.

Note Click OK to apply the new values to the page, tab, or Toolbox page.

The Page Order/Tab Order dialog box

To change the position of a page or tab:

1. Select the name of the **Page** or **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.

The 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.

The Customize Control dialog box

Contains the **ControlTipText** property and the icon that represents this control in the Toolbox. With this dialog box, you can define or change the **ControlTipText** associated with this control, as well as change the icon that is displayed in the Toolbox.

To define or edit **ControlTipText**:

- Enter a new value for **ControlTipText**.

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.

Item Method Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpItemC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpItemA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpItemS"}

The following example uses the **Item** method to access individual members of the **Controls** and **Pages** collections. The user chooses an option button for either the **Controls** collection or the **MultiPage**, and then clicks the **CommandButton**. The name of the appropriate control is returned in the **Label**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **CommandButton** named CommandButton1.
- A **Label** named Label1.
- Two **OptionButton** controls named OptionButton1 and OptionButton2.
- A **MultiPage** named MultiPage1.

```
Dim MyControl As Object
Dim ControlsIndex As Integer

Private Sub CommandButton1_Click()
    If OptionButton1.Value = True Then
        'Process Controls collection for UserForm
        Set MyControl = Controls.Item(ControlsIndex)
        Label1.Caption = MyControl.Name

        'Prepare index for next control on Userform
        ControlsIndex = ControlsIndex + 1
        If ControlsIndex >= Controls.Count Then
            ControlsIndex = 0
        End If

    ElseIf OptionButton2.Value = True Then
        'Process Current Page of Pages collection
        Set MyControl = MultiPage1.Pages.Item(MultiPage1.Value)
        Label1.Caption = MyControl.Name
    End If
End Sub

Private Sub UserForm_Initialize()
    ControlsIndex = 0
    'TabsIndex = 0

    OptionButton1.Caption = "Controls Collection"
    OptionButton2.Caption = "Pages Collection"
    OptionButton1.Value = True

    CommandButton1.Caption = "Get Member Name"
End Sub
```

Object Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpObjectC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpObjectA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpObjectS"}

Assume a new control has a **Top** property that is different from the standard **Top** property in Microsoft Forms. You can use either property, based on the syntax:

- `control.Top` uses the standard **Top** property.
- `control.Object.Top` uses the **Top** property from the added control.

TabStop Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpTabStopC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpTabStopA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpTabStopS"}

The following example uses the **TabStop** property to control whether a user can press TAB to move the focus to a particular control. The user presses TAB to move the focus among the controls on the form, and then clicks the **ToggleButton** to change **TabStop** for CommandButton1. When **TabStop** is **False**, CommandButton1 will not receive the focus by using TAB.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **CommandButton** named CommandButton1.
- A **ToggleButton** named ToggleButton1.
- One or two other controls, such as an **OptionButton** or **ListBox**.

```
Private Sub CommandButton1_Click()
    MsgBox "Clicked CommandButton1."
End Sub

Private Sub ToggleButton1_Click()
    If ToggleButton1 = True Then
        CommandButton1.TabStop = True
        ToggleButton1.Caption = "TabStop On"
    Else
        CommandButton1.TabStop = False
        ToggleButton1.Caption = "TabStop Off"
    End If
End Sub

Private Sub UserForm_Initialize()
    CommandButton1.Caption = "Show Message"

    ToggleButton1.Caption = "TabStop On"
    ToggleButton1.Value = True
    ToggleButton1.Width = 90
End Sub
```

TakeFocusOnClick Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpTakeFocusOnClickC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpTakeFocusOnClickA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpTakeFocusOnClickS"}

The following example uses the **TakeFocusOnClick** property to control whether a **CommandButton** receives the focus when the user clicks on it. The user clicks a control other than CommandButton1 and then clicks CommandButton1. If **TakeFocusOnClick** is **True**, CommandButton1 receives the focus after it is clicked. The user can change the value of **TakeFocusOnClick** by clicking the **ToggleButton**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **CommandButton** named CommandButton1.
- A **ToggleButton** named ToggleButton1.
- One or two other controls, such as an **OptionButton** or **ListBox**.

```
Private Sub CommandButton1_Click()
    MsgBox "Watch CommandButton1 to see if it takes the focus."
End Sub

Private Sub ToggleButton1_Click()
    If ToggleButton1 = True Then
        CommandButton1.TakeFocusOnClick = True
        ToggleButton1.Caption = "TakeFocusOnClick On"
    Else
        CommandButton1.TakeFocusOnClick = False
        ToggleButton1.Caption = "TakeFocusOnClick Off"
    End If
End Sub

Private Sub UserForm_Initialize()
    CommandButton1.Caption = "Show Message"

    ToggleButton1.Caption = "TakeFocusOnClick On"
    ToggleButton1.Value = True
    ToggleButton1.Width = 90
End Sub
```

MatchEntry Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpMatchEntryC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpMatchEntryA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpMatchEntryS"}

The following example uses the **MatchEntry** property to demonstrate character matching that is available for **ComboBox** and **ListBox**. In this example, the user can set the type of matching with the **OptionButton** controls and then type into the **ComboBox** to specify an item from its list.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- Three **OptionButton** controls named OptionButton1 through OptionButton3.
- A **ComboBox** named ComboBox1.

```
Private Sub OptionButton1_Click()
    ComboBox1.MatchEntry = fmMatchEntryNone
End Sub

Private Sub OptionButton2_Click()
    ComboBox1.MatchEntry = fmMatchEntryFirstLetter
End Sub

Private Sub OptionButton3_Click()
        ComboBox1.MatchEntry = fmMatchEntryComplete
End Sub

Private Sub UserForm_Initialize()
    Dim i As Integer

    For i = 1 To 9
        ComboBox1.AddItem "Choice " & i
    Next i
    ComboBox1.AddItem "Chocoholic"


    OptionButton1.Caption = "No matching"
    OptionButton1.Value = True

    OptionButton2.Caption = "Basic matching"
    OptionButton3.Caption = "Extended matching"
End Sub
```

MatchFound, MatchRequired Properties Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpMatchFoundMatchRequiredC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpMatchFoundMatchRequiredA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpMatchFoundMatchRequiredS"}

The following example uses the **MatchFound** and **MatchRequired** properties to demonstrate additional character matching for **ComboBox**. The matching verification occurs in the Change event.

In this example, the user specifies whether the text portion of a **ComboBox** must match one of the listed items in the **ComboBox**. The user can specify whether matching is required by using a **CheckBox** and then type into the **ComboBox** to specify an item from its list.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **ComboBox** named ComboBox1.
- A **CheckBox** named CheckBox1.

```
Private Sub CheckBox1_Click()
    If CheckBox1.Value = True Then
        ComboBox1.MatchRequired = True
        MsgBox "To move the focus from the ComboBox, you must match an
entry in the list or press ESC."
    Else
        ComboBox1.MatchRequired = False
        MsgBox " To move the focus from the ComboBox, just tab to or click
another control. Matching is optional."
    End If
End Sub

Private Sub ComboBox1_Change()
    If ComboBox1.MatchRequired = True Then
    'MSForms handles this case automatically
    Else
        If ComboBox1.MatchFound = True Then
            MsgBox "Match Found; matching optional."
        Else
            MsgBox "Match not Found; matching optional."
        End If
    End If
End Sub

Private Sub UserForm_Initialize()
Dim i As Integer

For i = 1 To 9
    ComboBox1.AddItem "Choice " & i
Next i
ComboBox1.AddItem "Chocoholic"

CheckBox1.Caption = "MatchRequired"
CheckBox1.Value = True
End Sub
```

MultiSelect, Selected Properties Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpMultiSelectSelectedC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpMultiSelectSelectedA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpMultiSelectSelectedS"}

The following example uses the **MultiSelect** and **Selected** properties to demonstate how the user can select one or more items in a **ListBox**. The user specifies a selection method by choosing an option button and then selects an item(s) from the **ListBox**. The user can display the selected items in a second **ListBox** by clicking the **CommandButton**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- Two **ListBox** controls named ListBox1 and ListBox2.
- A **CommandButton** named CommandButton1.
- Three **OptionButton** controls named OptionButton1 through OptionButton3.

```
Dim i As Integer

Private Sub CommandButton1_Click()
    ListBox2.Clear

    For i = 0 To 9
        If ListBox1.Selected(i) = True Then
            ListBox2.AddItem ListBox1.List(i)
        End If
    Next i

End Sub

Private Sub OptionButton1_Click()
    ListBox1.MultiSelect = fmMultiSelectSingle
End Sub

Private Sub OptionButton2_Click()
    ListBox1.MultiSelect = fmMultiSelectMulti
End Sub

Private Sub OptionButton3_Click()
    ListBox1.MultiSelect = fmMultiSelectExtended
End Sub

Private Sub UserForm_Initialize()
    For i = 0 To 9
        ListBox1.AddItem "Choice " & (ListBox1.ListCount + 1)
    Next i

    OptionButton1.Caption = "Single Selection"
    ListBox1.MultiSelect = fmMultiSelectSingle
    OptionButton1.Value = True

    OptionButton2.Caption = "Multiple Selection"
    OptionButton3.Caption = "Extended Selection"

    CommandButton1.Caption = "Show selections"
    CommandButton1.AutoSize = True
End Sub
```

Style Property Example for ComboBox

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpStyleComboBoxC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpStyleComboBoxA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpStyleComboBoxS"}

The following example uses the **Style** property to change the effect of typing in the text area of a **ComboBox**. The user chooses a style by selecting an **OptionButton** control and then types into the **ComboBox** to select an item. When **Style** is *fmStyleDropDownList*, the user must choose an item from the drop-down list. When **Style** is *fmStyleDropDownCombo*, the user can type into the text area to specify an item in the drop-down list.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- Two **OptionButton** controls named OptionButton1 and OptionButton2.
- A **ComboBox** named ComboBox1.

```
Private Sub OptionButton1_Click()
    ComboBox1.Style = fmStyleDropDownCombo
End Sub

Private Sub OptionButton2_Click()
    ComboBox1.Style = fmStyleDropDownList
End Sub

Private Sub UserForm_Initialize()
    Dim i As Integer

    For i = 1 To 10
        ComboBox1.AddItem "Choice " & i
    Next i

    OptionButton1.Caption = "Select like ComboBox"
    OptionButton1.Value = True
    ComboBox1.Style = fmStyleDropDownCombo

    OptionButton2.Caption = "Select like ListBox"
End Sub
```

Style Property Example for MultiPage and TabStrip

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpStyleMultiPageTabStripC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpStyleMultiPageTabStripA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpStyleMultiPageTabStripS"}

The following example uses the **Style** property to specify the appearance of the tabs in **MultiPage** and **TabStrip**. This example also demonstrates using a **Label**. The user chooses a style by selecting an **OptionButton**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **Label** named Label1.
- Three **OptionButton** controls named OptionButton1 through OptionButton3.
- A **MultiPage** named MultiPage1.
- A **TabStrip** named TabStrip1.
- Any control inside the **TabStrip**.
- Any control in each page of the **MultiPage**.

```
Private Sub OptionButton1_Click()
    MultiPage1.Style = fmTabStyleTabs
    TabStrip1.Style = fmTabStyleTabs
End Sub

Private Sub OptionButton2_Click()
    'Note that the page borders are invisible
    MultiPage1.Style = fmTabStyleButtons
    TabStrip1.Style = fmTabStyleButtons
End Sub

Private Sub OptionButton3_Click()
    'Note that the page borders are invisible and
    'the page body begins where the tabs normally appear.
    MultiPage1.Style = fmTabStyleNone
    TabStrip1.Style = fmTabStyleNone
End Sub

Private Sub UserForm_Initialize()
    Label1.Caption = "Page/Tab Style"
    OptionButton1.Caption = "Tabs"
    OptionButton1.Value = True
    MultiPage1.Style = fmTabStyleTabs
    TabStrip1.Style = fmTabStyleTabs

    OptionButton2.Caption = "Buttons"
    OptionButton3.Caption = "No Tabs or Buttons"
End Sub
```

OldLeft, OldTop, OldHeight, OldWidth Properties Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpOldLeftOldTopOldHeightOldWidthC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpOldLeftOldTopOldHeightOldWidthA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpOldLeftOldTopOldHeightOldWidthS"}

The following example uses the **OldLeft**, **OldTop**, **OldHeight**, and **OldWidth** properties within the **Layout** event to keep a control at its current position and size. The user clicks the **CommandButton** labeled **Move ComboBox** to move the control, and then responds to a message box. The user can click the **CommandButton** labeled **Reset ComboBox** to reset the control for another repetition.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- Two **CommandButton** controls named CommandButton1 and CommandButton2.
- A **ComboBox** named ComboBox1.

```
Dim Initialize As Integer
Dim ComboLeft, ComboTop, ComboWidth, ComboHeight As Integer

Private Sub UserForm_Initialize()
    Initialize = 0
    CommandButton1.Caption = "Move ComboBox"
    CommandButton2.Caption = "Reset ComboBox"

    'Information for resetting ComboBox
    ComboLeft = ComboBox1.Left
    ComboTop = ComboBox1.Top
    ComboWidth = ComboBox1.Width
    ComboHeight = ComboBox1.Height
End Sub

Private Sub CommandButton1_Click()
        ComboBox1.Move 0, 0, , , True
End Sub

Private Sub UserForm_Layout()
    Dim MyControl As Control
    Dim MsgBoxResult As Integer

    If Initialize = 0 Then   'Suppress MsgBox on initial layout event.
        Initialize = 1
        Exit Sub
    End If

    MsgBoxResult = MsgBox("In Layout event - Continue move?", vbYesNo)
    If MsgBoxResult = vbNo Then
        ComboBox1.Move ComboBox1.OldLeft, ComboBox1.OldTop,
ComboBox1.OldWidth, ComboBox1.OldHeight
    End If
End Sub

Private Sub CommandButton2_Click()
    ComboBox1.Move ComboLeft, ComboTop, ComboWidth, ComboHeight

    'OldLeft, OldTop, OldWidth, and OldHeight are not recognized here.
    'The following statement, if not commented, would produce an error at
run time.
```

```
    'ComboBox1.Move ComboBox1.OldLeft, ComboBox1.OldTop,
ComboBox1.OldWidth, ComboBox1.OldHeight
End Sub
```

TabFixedHeight, TabFixedWidth Properties Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpTabFixedHeightTabFixedWidthC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpTabFixedHeightTabFixedWidthA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpTabFixedHeightTabFixedWidthS"}

The following example uses the **TabFixedHeight** and **TabFixedWidth** properties to set the size of the tabs used in **MultiPage** and **TabStrip**. The user clicks the **SpinButton** controls to adjust the height and width of the tabs within the **MultiPage** and **TabStrip**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **MultiPage** named MultiPage1.
- A **TabStrip** named TabStrip1.
- A **Label** named Label1 for the width control.
- A **SpinButton** named SpinButton1 for the width control.
- A **TextBox** named TextBox1 for the width control.
- A **Label** named Label2 for the height control.
- A **SpinButton** named SpinButton2 for the height control.
- A **TextBox** named TextBox2 for the height control.

```
Private Sub UpdateTabWidth()
    TextBox1.Text = SpinButton1.Value
    TabStrip1.TabFixedWidth = SpinButton1.Value
    MultiPage1.TabFixedWidth = SpinButton1.Value
End Sub

Private Sub UpdateTabHeight()
    TextBox2.Text = SpinButton2.Value
    TabStrip1.TabFixedHeight = SpinButton2.Value
    MultiPage1.TabFixedHeight = SpinButton2.Value
End Sub

Private Sub UserForm_Initialize()
    MultiPage1.Style = fmTabStyleButtons

    Label1.Caption = "Tab Width"
    SpinButton1.Min = 0
    SpinButton1.Max = TabStrip1.Width / TabStrip1.Tabs.Count
    SpinButton1.Value = 0
    TextBox1.Locked = True

    UpdateTabWidth

    Label2.Caption = "Tab Height"
    SpinButton2.Min = 0
    SpinButton2.Max = TabStrip1.Height
    SpinButton2.Value = 0
    TextBox2.Locked = True

    UpdateTabHeight
End Sub

Private Sub SpinButton1_SpinDown()
    UpdateTabWidth
End Sub
```

```
Private Sub SpinButton1_SpinUp()
    UpdateTabWidth
End Sub

Private Sub SpinButton2_SpinDown()
    UpdateTabHeight
End Sub

Private Sub SpinButton2_SpinUp()
    UpdateTabHeight
End Sub
```

TabIndex Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpTabIndexC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpTabIndexA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpTabIndexS"}

The following example uses the **TabIndex** property to display and set the tab order for individual controls. The user can press TAB to reach the next control in the tab order and to display the **TabIndex** of that control. The user can also click on a control to display its **TabIndex**. The User can change the **TabIndex** of a control by specifying a new index value in the **TextBox** and clicking CommandButton3. Changing the **TabIndex** for one control also updates the **TabIndex** for other controls in the **Frame**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **Label** named Label1.
- A **TextBox** named TextBox1.
- A **Frame** named Frame1.
- A **TextBox** in the **Frame** named TextBox2.
- Two **CommandButton** controls in the **Frame** named CommandButton1 and CommandButton2.
- A **ScrollBar** in the **Frame** named ScrollBar1.
- A **CommandButton** (not in the **Frame**) named CommandButton3.

```
Private Sub MoveToFront()
    Dim i, Temp As Integer

    Temp = Frame1.ActiveControl.TabIndex
    For i = 0 To Temp - 1
        Frame1.Controls.Item(i).TabIndex = i + 1
    Next i

    Frame1.ActiveControl.TabIndex = 0
    TextBox1.Text = Frame1.ActiveControl.TabIndex
End Sub

Private Sub CommandButton3_Click()
    Dim i, Temp As Integer

    If IsNumeric(TextBox1.Text) Then
        Temp = Val(TextBox1.Text)

        If Temp >= Frame1.Controls.Count Or Temp < 0 Then
            'Entry out of range; move control to front of tab order
            MoveToFront
        ElseIf Temp > Frame1.ActiveControl.TabIndex Then
            'Move entry down the list
            For i = Frame1.ActiveControl.TabIndex + 1 To Temp
                Frame1.Controls.Item(i).TabIndex = i - 1
            Next i
            Frame1.ActiveControl.TabIndex = Temp
            TextBox1.Text = Frame1.ActiveControl.TabIndex
        Else
            'Move Entry up the list
            For i = Frame1.ActiveControl.TabIndex - 1 To Temp
                Frame1.Controls.Item(i).TabIndex = i + 1
            Next i
            Frame1.ActiveControl.TabIndex = Temp
```

```
            TextBox1.Text = Frame1.ActiveControl.TabIndex
        End If
    Else
        'Text entry; move control to front of tab order
        MoveToFront
    End If
End Sub

Private Sub UserForm_Initialize()
    Label1.Caption = "TabIndex"

    Frame1.Controls(0).SetFocus
    TextBox1.Text = Frame1.ActiveControl.TabIndex

    Frame1.Cycle = fmCycleCurrentForm

    CommandButton3.Caption = "Set TabIndex"
    CommandButton3.TakeFocusOnClick = False
End Sub

Private Sub TextBox2_Enter()
    TextBox1.Text = Frame1.ActiveControl.TabIndex
End Sub

Private Sub CommandButton1_Enter()
    TextBox1.Text = Frame1.ActiveControl.TabIndex
End Sub

Private Sub CommandButton2_Enter()
    TextBox1.Text = Frame1.ActiveControl.TabIndex
End Sub

Private Sub ScrollBar1_Enter()
    TextBox1.Text = Frame1.ActiveControl.TabIndex
End Sub
```

Layout Event, LayoutEffect Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpLayoutLayoutEffectC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpLayoutLayoutEffectA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpLayoutLayoutEffectS"}

The following example moves a selected control on a form with the **Move** method, and uses the Layout event and **LayoutEffect** property to identify the control that moved (and changed the layout of the **UserForm**). The user clicks a control to move and then clicks the **CommandButton**. A message box displays the name of the control that is moving.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **TextBox** named TextBox1.
- A **ComboBox** named ComboBox1.
- An **OptionButton** named OptionButton1.
- A **CommandButton** named CommandButton1.
- A **ToggleButton** named ToggleButton1.

```
Private Sub UserForm_Initialize()
    CommandButton1.Caption = "Move current control"
    CommandButton1.AutoSize = True
    CommandButton1.TakeFocusOnClick = False

    ToggleButton1.Caption = "Use Layout Event"
    ToggleButton1.Value = True
End Sub

Private Sub CommandButton1_Click()
    If ActiveControl.Name = "ToggleButton1" Then
        'Keep it stationary
    Else
        'Move the control, using Layout event when ToggleButton1.Value is
True
        ActiveControl.Move 0, 0, , , ToggleButton1.Value
    End If
End Sub

Private Sub UserForm_Layout()
    Dim MyControl As Control

    MsgBox "In the Layout Event"

    'Find the control that is moving.
    For Each MyControl In Controls
        If MyControl.LayoutEffect = fmLayoutEffectInitiate Then
            MsgBox MyControl.Name & " is moving."
            Exit For
        End If
    Next
End Sub

Private Sub ToggleButton1_Click()
    If ToggleButton1.Value = True Then
        ToggleButton1.Caption = "Use Layout Event"
    Else
        ToggleButton1.Caption = "No Layout Event"
```

```
    End If
End Sub
```

Tag Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpTagC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpTagA"}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpTagS"}

The following example uses the **Tag** property to store additional information about each control on the **UserForm**. The user clicks a control and then clicks the **CommandButton**. The contents of **Tag** for the appropriate control are returned in the **TextBox**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **TextBox** named TextBox1.
- A **CommandButton** named CommandButton1.
- A **ScrollBar** named ScrollBar1.
- A **ComboBox** named ComboBox1.
- A **MultiPage** named MultiPage1.

```
Private Sub CommandButton1_Click()
    TextBox1.Text = ActiveControl.Tag
End Sub

Private Sub UserForm_Initialize()
    TextBox1.Locked = True
    TextBox1.Tag = "Display area for Tag properties."
    TextBox1.AutoSize = True

    CommandButton1.Caption = "Show Tag of Current Control."
    CommandButton1.AutoSize = True
    CommandButton1.WordWrap = True
    CommandButton1.TakeFocusOnClick = False
    CommandButton1.Tag = "Shows tag of control that has the focus."

    ComboBox1.Style = fmStyleDropDownList
    ComboBox1.Tag = "ComboBox Style is that of a ListBox."

    ScrollBar1.Max = 100
    ScrollBar1.Min = -273
    ScrollBar1.Tag = "Max = " & ScrollBar1.Max & " , Min = " &
ScrollBar1.Min

    MultiPage1.Pages.Add
    MultiPage1.Pages.Add
    MultiPage1.Tag = "This MultiPage has " & MultiPage1.Pages.Count & "
pages."
End Sub
```

TopIndex Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpTopIndexC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpTopIndexA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpTopIndexS"}

The following example identifies the top item displayed in a **ListBox** and the item that has the focus within the **ListBox**. This example uses the **TopIndex** property to identify the item displayed at the top of the **ListBox** and the **ListIndex** property to identify the item that has the focus. The user selects an item in the **ListBox**. The displayed values of **TopIndex** and **ListIndex** are updated when the user selects an item or when the user clicks the **CommandButton**..

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **Label** named Label1.
- A **TextBox** named TextBox1.
- A **Label** named Label2.
- A **TextBox** named TextBox2.
- A **CommandButton** named CommandButton1.
- A **ListBox** named ListBox1.

```
Private Sub CommandButton1_Click()
    ListBox1.TopIndex = ListBox1.ListIndex
    TextBox1.Text = ListBox1.TopIndex
    TextBox2.Text = ListBox1.ListIndex
End Sub

Private Sub ListBox1_Change()
    TextBox1.Text = ListBox1.TopIndex
    TextBox2.Text = ListBox1.ListIndex
End Sub

Private Sub UserForm_Initialize()
    Dim i As Integer

    For i = 0 To 24
        ListBox1.AddItem "Choice " & (i + 1)
    Next i
    ListBox1.Height = 66
    CommandButton1.Caption = "Move to top of list"
    CommandButton1.AutoSize = True
    CommandButton1.TakeFocusOnClick = False

    Label1.Caption = "Index of top item"
    TextBox1.Text = ListBox1.TopIndex

    Label2. Caption = "Index of current item"
    Label2.AutoSize = True
    Label2.WordWrap = False
    TextBox2.Text = ListBox1.ListIndex
End Sub
```

TripleState Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpTripleStateC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpTripleStateA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpTripleStateS"}

The following example uses the **TripleState** property to allow **Null** as a legal value of a **CheckBox** and a **ToggleButton**. The user controls the value of **TripleState** through ToggleButton2. The user can set the value of a **CheckBox** or **ToggleButton** based on the value of **TripleState**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **CheckBox** named CheckBox1.
- A **ToggleButton** named ToggleButton1.
- A **ToggleButton** named ToggleButton2.

```
Private Sub UserForm_Initialize()
    CheckBox1.Caption = "Value is True"
    CheckBox1.Value = True
    CheckBox1.TripleState = False

    ToggleButton1.Caption = "Value is True"
    ToggleButton1.Value = True
    ToggleButton1.TripleState = False

    ToggleButton2.Value = False
    ToggleButton2.Caption = "Triple State Off"
End Sub

Private Sub ToggleButton2_Click()
    If ToggleButton2.Value = True Then
        ToggleButton2.Caption = "Triple State On"
        CheckBox1.TripleState = True
        ToggleButton1.TripleState = True
    Else
        ToggleButton2.Caption = "Triple State Off"
        CheckBox1.TripleState = False
        ToggleButton1.TripleState = False
    End If
End Sub

Private Sub CheckBox1_Change()
    If IsNull(CheckBox1.Value) Then
        CheckBox1.Caption = "Value is Null"
    ElseIf CheckBox1.Value = False Then
        CheckBox1.Caption = "Value is False"
    ElseIf CheckBox1.Value = True Then
        CheckBox1.Caption = "Value is True"
    End If
End Sub

Private Sub ToggleButton1_Change()
    If IsNull(ToggleButton1.Value) Then
        ToggleButton1.Caption = "Value is Null"
    ElseIf ToggleButton1.Value = False Then
        ToggleButton1.Caption = "Value is False"
    ElseIf ToggleButton1.Value = True Then
```

```
        ToggleButton1.Caption = "Value is True"
    End If
End Sub
```

Value Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpValueC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpValueA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpValueS"}

The following example demonstrates the values that the different types of controls can have by displaying the **Value** property of a selected control. The user chooses a control by pressing TAB or by clicking on the control. Depending on the type of control, the user can also specify a value for the control by typing in the text area of the control, by clicking one or more times on the control, or by selecting an item, page, or tab within the control. The user can display the value of the selected control by clicking the appropriately labeled **CommandButton**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **CommandButton** named CommandButton1.
- A **TextBox** named TextBox1.
- A **CheckBox** named CheckBox1.
- A **ComboBox** named ComboBox1.
- A **CommandButton** named CommandButton2.
- A **ListBox** named ListBox1.
- A **MultiPage** named MultiPage1.
- Two **OptionButton** controls named OptionButton1 and OptionButton2.
- A **ScrollBar** named ScrollBar1.
- A **SpinButton** named SpinButton1.
- A **TabStrip** named TabStrip1.
- A **TextBox** named TextBox2.
- A **ToggleButton** named ToggleButton1.

```
Dim i As Integer

Private Sub CommandButton1_Click()
    TextBox1.Text = "Value of " & ActiveControl.Name & " is " &
ActiveControl.Value
End Sub

Private Sub UserForm_Initialize()
    CommandButton1.Caption = "Get value of current control"
    CommandButton1.AutoSize = True
    CommandButton1.TakeFocusOnClick = False
    CommandButton1.TabStop = False

    TextBox1.AutoSize = True

    For i = 0 To 10
        ComboBox1.AddItem "Choice " & (i + 1)
        ListBox1.AddItem "Selection " & (100 - i)
    Next i

    CheckBox1.TripleState = True
    ToggleButton1.TripleState = True

    TextBox2.Text = "Enter text here."
End Sub
```

KeyPress Event Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpKeyPressC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpKeyPressA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpKeyPressS"}

The following example uses the **KeyPress** event to copy keystrokes from one **TextBox** to a second **TextBox**. The user types into the appropriately marked **TextBox**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- Two **TextBox** controls named TextBox1 and TextBox2.

```
Private Sub TextBox1_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
    TextBox2.Text = TextBox2.Text & Chr(KeyAscii)

    'To handle keyboard combinations (using SHIFT, CTRL, ALT, and another
key),
    'or TAB or ENTER, use the KeyDown or KeyUp event.
End Sub

Private Sub UserForm_Initialize()
    Move 0, 0, 570, 380

    TextBox1.Move 30, 40, 220, 160
    TextBox1.MultiLine = True
    TextBox1.WordWrap = True
    TextBox1.Text = "Type text here."
    TextBox1.EnterKeyBehavior = True


    TextBox2.Move 298, 40, 220, 160
    TextBox2.MultiLine = True
    TextBox2.WordWrap = True
    TextBox2.Text = "Typed text copied here."
    TextBox2.Locked = True
 End Sub
```

Zoom Event Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpZoomEventC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpZoomEventA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpZoomEventS"}

The following example uses the **Zoom** event to evaluate the new value of the **Zoom** property and adds scroll bars to the form when appropriate. The example uses a **Label** to display the current value. The user specifies the size for the form by using the **SpinButton** and then clicks the **CommandButton** to set the value in the **Zoom** property.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **Label** named Label1.
- A **SpinButton** named SpinButton1.
- A **CommandButton** named CommandButton1.
- Other controls placed near the edges of the form.

```
Private Sub CommandButton1_Click()
    Zoom = SpinButton1.Value
End Sub

Private Sub SpinButton1_SpinDown()
    Label1.Caption = SpinButton1.Value
End Sub

Private Sub SpinButton1_SpinUp()
    Label1.Caption = SpinButton1.Value
End Sub

Private Sub UserForm_Initialize()
    SpinButton1.Min = 10
    SpinButton1.Max = 400
    SpinButton1.Value = 100
    Label1.Caption = SpinButton1.Value

    CommandButton1.Caption = "Zoom it!"
End Sub

Private Sub UserForm_Zoom(Percent As Integer)
    Dim MyResult As Double

    If Percent > 99 Then
        ScrollBars = fmScrollBarsBoth
        ScrollLeft = 0
        ScrollTop = 0

        MyResult = Width * Percent / 100
        ScrollWidth = MyResult

        MyResult = Height * Percent / 100
        ScrollHeight = MyResult
    Else
        ScrollBars = fmScrollBarsNone
        ScrollLeft = 0
        ScrollTop = 0
```

```
    End If
End Sub
```

Max, Min, MaxLength Properties Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpMaxMinMaxLengthC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpMaxMinMaxLengthA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpMaxMinMaxLengthS"}

The following example demonstrates the **Max** and **Min** properties when used with a stand-alone **ScrollBar**. The user can set the **Max** and **Min** values to any integer in the range of –1000 to 1000. This example also uses the **MaxLength** property to restrict the number of characters entered for the **Max** and **Min** values.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **Label** named Label1 and a **TextBox** named TextBox1.
- A **Label** named Label2 and a **TextBox** named TextBox2.
- A **ScrollBar** named ScrollBar1.
- A **Label** named Label3.

```
Dim TempNum As Integer

Private Sub UserForm_Initialize()
    Label1.Caption = "Min -1000 to 1000"
    ScrollBar1.Min = -1000
    TextBox1.Text = ScrollBar1.Min
    TextBox1.MaxLength = 5

    Label2.Caption = "Max -1000 to 1000"
    ScrollBar1.Max = 1000
    TextBox2.Text = ScrollBar1.Max
    TextBox2.MaxLength = 5

    ScrollBar1.SmallChange = 1
    ScrollBar1.LargeChange = 100
    ScrollBar1.Value = 0
    Label3.Caption = ScrollBar1.Value
End Sub

Private Sub TextBox1_Change()
    If IsNumeric(TextBox1.Text) Then
        TempNum = CInt(TextBox1.Text)
        If TempNum >= -1000 And TempNum <= 1000 Then
            ScrollBar1.Min = TempNum
        Else
            TextBox1.Text = ScrollBar1.Min
        End If
    Else
        TextBox1.Text = ScrollBar1.Min
    End If
End Sub

Private Sub TextBox2_Change()
    If IsNumeric(TextBox2.Text) Then
        TempNum = CInt(TextBox2.Text)
        If TempNum >= -1000 And TempNum <= 1000 Then
            ScrollBar1.Max = TempNum
        Else
            TextBox2.Text = ScrollBar1.Max
```

```
        End If
    Else
        TextBox2.Text = ScrollBar1.Max
    End If
End Sub

Private Sub ScrollBar1_Change()
Label3.Caption = ScrollBar1.Value
End Sub
```

LargeChange, SmallChange Properties Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpLargeChangeSmallChangeC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpLargeChangeSmallChangeA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpLargeChangeSmallChangeS"}

The following example demonstrates the **LargeChange** and **SmallChange** properties when used with a stand-alone **ScrollBar**. The user can set the **LargeChange** and **SmallChange** values to any integer in the range of 0 to 100. This example also uses the **MaxLength** property to restrict the number of characters entered for the **LargeChange** and **SmallChange** values.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **Label** named Label1 and a **TextBox** named TextBox1.
- A **Label** named Label2 and a **TextBox** named TextBox2.
- A **ScrollBar** named ScrollBar1.
- A **Label** named Label3.

```
Dim TempNum As Integer

Private Sub ScrollBar1_Change()
    Label3.Caption = ScrollBar1.Value
End Sub

Private Sub TextBox1_Change()
    If IsNumeric(TextBox1.Text) Then
        TempNum = CInt(TextBox1.Text)
        If TempNum >= 0 And TempNum <= 100 Then
            ScrollBar1.SmallChange = TempNum
        Else
            TextBox1.Text = ScrollBar1.SmallChange
        End If
    Else
        TextBox1.Text = ScrollBar1.SmallChange
    End If
End Sub

Private Sub TextBox2_Change()
    If IsNumeric(TextBox2.Text) Then
        TempNum = CInt(TextBox2.Text)
        If TempNum >= 0 And TempNum <= 100 Then
            ScrollBar1.LargeChange = TempNum
        Else
            TextBox2.Text = ScrollBar1.LargeChange
        End If
    Else
        TextBox2.Text = ScrollBar1.LargeChange
    End If
End Sub

Private Sub UserForm_Initialize()
    ScrollBar1.Min = -1000
    ScrollBar1.Max = 1000

    Label1.Caption = "SmallChange 0 to 100"
    ScrollBar1.SmallChange = 1
    TextBox1.Text = ScrollBar1.SmallChange
```

```
    TextBox1.MaxLength = 3

    Label2.Caption = "LargeChange 0 to 100"
    ScrollBar1.LargeChange = 100
    TextBox2.Text = ScrollBar1.LargeChange
    TextBox2.MaxLength = 3

    ScrollBar1.Value = 0
    Label3.Caption = ScrollBar1.Value
End Sub
```

ScrollBars, KeepScrollBarsVisible Properties Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpScrollBarsKeepScrollBarsVisibleC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpScrollBarsKeepScrollBarsVisibleA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpScrollBarsKeepScrollBarsVisibleS"}

The following example uses the **ScrollBars** and the **KeepScrollBarsVisible** properties to add scroll bars to a page of a **MultiPage** and to a **Frame**. The user chooses an option button that, in turn, specifies a value for **KeepScrollBarsVisible**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **MultiPage** named MultiPage1.
- A **Frame** named Frame1.
- Four **OptionButton** controls named OptionButton1 through OptionButton4.

```
Private Sub UserForm_Initialize()
    MultiPage1.Pages(0).ScrollBars = fmScrollBarsBoth
    MultiPage1.Pages(0).KeepScrollBarsVisible = fmScrollBarsNone

    Frame1.ScrollBars = fmScrollBarsBoth
    Frame1.KeepScrollBarsVisible = fmScrollBarsNone

    OptionButton1.Caption = "No scroll bars"
    OptionButton1.Value = True
    OptionButton2.Caption = "Horizontal scroll bars"
    OptionButton3.Caption = "Vertical scroll bars"
    OptionButton4.Caption = "Both scroll bars"
End Sub

Private Sub OptionButton1_Click()
    MultiPage1.Pages(0).KeepScrollBarsVisible = fmScrollBarsNone
    Frame1.KeepScrollBarsVisible = fmScrollBarsNone
End Sub

Private Sub OptionButton2_Click()
    MultiPage1.Pages(0).KeepScrollBarsVisible = fmScrollBarsHorizontal
    Frame1.KeepScrollBarsVisible = fmScrollBarsHorizontal
End Sub

Private Sub OptionButton3_Click()
    MultiPage1.Pages(0).KeepScrollBarsVisible = fmScrollBarsVertical
    Frame1.KeepScrollBarsVisible = fmScrollBarsVertical
End Sub

Private Sub OptionButton4_Click()
    MultiPage1.Pages(0).KeepScrollBarsVisible = fmScrollBarsBoth
    Frame1.KeepScrollBarsVisible = fmScrollBarsBoth
End Sub
```

ScrollHeight, ScrollLeft, ScrollTop, ScrollWidth Properties Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpScrollHeightC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpScrollHeightA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpScrollHeightS"}

The following example uses a page of a **MultiPage** as a scrolling region. The user can use the scroll bars on Page2 of the **MultiPage** to gain access to parts of the page that are not initially displayed.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains a **MultiPage** named MultiPage1, and that each page of the **MultiPage** contains one or more controls.

Note Each page of a **MultiPage** is unique. Page1 has no scroll bars. Page2 has horizontal and vertical scroll bars.

```
Private Sub UserForm_Initialize()
    MultiPage1.Pages(1).ScrollBars = fmScrollBarsBoth
    MultiPage1.Pages(1).KeepScrollBarsVisible = fmScrollBarsNone

    MultiPage1.Pages(1).ScrollHeight = 2 * MultiPage1.Height
    MultiPage1.Pages(1).ScrollWidth = 2 * MultiPage1.Width

    'Set ScrollHeight, ScrollWidth before setting ScrollLeft, ScrollTop
    MultiPage1.Pages(1).ScrollLeft = MultiPage1.Width / 2
    MultiPage1.Pages(1).ScrollTop = MultiPage1.Height / 2
End Sub
```

InsideHeight, InsideWidth Properties Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpInsideHeightInsideWidthC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpInsideHeightInsideWidthA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpInsideHeightInsideWidthS"}

The following example uses the **InsideHeight** and **InsideWidth** properties to resize a **CommandButton**. The user clicks the **CommandButton** to resize it.

Note **InsideHeight** and **InsideWidth** are read-only properties.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **CommandButton** named CommandButton1.

```
Dim Resize As Single

Private Sub UserForm_Initialize()
    Resize = 0.75
    CommandButton1.Caption = "Resize Button"

End Sub

Private Sub CommandButton1_Click()
    CommandButton1.Move 10, 10, UserForm1.InsideWidth * Resize,
UserForm1.InsideHeight * Resize
    CommandButton1.Caption = "Button resized using InsideHeight and
InsideWidth!"
End Sub
```

ListRows Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpListRowsC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpListRowsA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpListRowsS"}

The following example uses a **SpinButton** to control the number of rows in the drop-down list of a **ComboBox**. The user changes the value of the **SpinButton**, then clicks on the drop-down arrow of the **ComboBox** to display the list.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **ComboBox** named ComboBox1.
- A **SpinButton** named SpinButton1.
- A **Label** named Label1.

```
Private Sub UserForm_Initialize()
    Dim i As Integer

    For i = 1 To 20
        ComboBox1.AddItem "Choice " & (ComboBox1.ListCount + 1)
    Next i

    SpinButton1.Min = 0
    SpinButton1.Max = 12
    SpinButton1.Value = ComboBox1.ListRows
    Label1.Caption = "ListRows = " & SpinButton1.Value
End Sub

Private Sub SpinButton1_Change()
    ComboBox1.ListRows = SpinButton1.Value
    Label1.Caption = "ListRows = " & SpinButton1.Value
End Sub
```

ListWidth Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpListWidthC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpListWidthA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpListWidthS"}

The following example uses a **SpinButton** to control the width of the drop-down list of a **ComboBox**. The user changes the value of the **SpinButton**, then clicks on the drop-down arrow of the **ComboBox** to display the list.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **ComboBox** named ComboBox1.
- A **SpinButton** named SpinButton1.
- A **Label** named Label1.

```
Private Sub SpinButton1_Change()
    ComboBox1.ListWidth = SpinButton1.Value
    Label1.Caption = "ListWidth = " & SpinButton1.Value
End Sub

Private Sub UserForm_Initialize()
    Dim i As Integer

    For i = 1 To 20
        ComboBox1.AddItem "Choice " & (ComboBox1.ListCount + 1)
    Next i

    SpinButton1.Min = 0
    SpinButton1.Max = 130
    SpinButton1.Value = Val(ComboBox1.ListWidth)
    SpinButton1.SmallChange = 5
    Label1.Caption = "ListWidth = " & SpinButton1.Value
End Sub
```

ListStyle, MultiSelect Properties Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpListStyleMultiSelectC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpListStyleMultiSelectA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpListStyleMultiSelectS"}

The following example uses the **ListStyle** and **MultiSelect** properties to control the appearance of a **ListBox**. The user chooses a value for **ListStyle** using the **ToggleButton** and chooses an **OptionButton** for one of the **MultiSelect** values. The appearance of the **ListBox** changes accordingly, as well as the selection behavior within the **ListBox**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **ListBox** named ListBox1.
- A **Label** named Label1.
- Three **OptionButton** controls named OptionButton1 through OptionButton3.
- A **ToggleButton** named ToggleButton1.

```
Private Sub UserForm_Initialize()
    Dim i As Integer

    For i = 1 To 8
        ListBox1.AddItem "Choice" & (ListBox1.ListCount + 1)
    Next i

    Label1.Caption = "MultiSelect Choices"
    Label1.AutoSize = True

    ListBox1.MultiSelect = fmMultiSelectSingle
    OptionButton1.Caption = "Single entry"
    OptionButton1.Value = True
    OptionButton2.Caption = "Multiple entries"
    OptionButton3.Caption = "Extended entries"

    ToggleButton1.Caption = "ListStyle - Plain"
    ToggleButton1.Value = True
    ToggleButton1.Width = 90
    ToggleButton1.Height = 30
End Sub

Private Sub OptionButton1_Click()
    ListBox1.MultiSelect = fmMultiSelectSingle
End Sub

Private Sub OptionButton2_Click()
    ListBox1.MultiSelect = fmMultiSelectMulti
End Sub

Private Sub OptionButton3_Click()
    ListBox1.MultiSelect = fmMultiSelectExtended
End Sub

Private Sub ToggleButton1_Click()
    If ToggleButton1.Value = True Then
        ToggleButton1.Caption = "Plain ListStyle"
        ListBox1.ListStyle = fmListStylePlain
    Else
```

```
        ToggleButton1.Caption = "OptionButton or CheckBox"
        ListBox1.ListStyle = fmListStyleOption
    End If
End Sub
```

MouseIcon, MousePointer Properties Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpMouseIconMousePointerC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpMouseIconMousePointerA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpMouseIconMousePointerS"}

The following example demonstrates how to specify a mouse pointer that is appropriate for a specific control or situation. You can assign one of several available mouse pointers using the **MousePointer** property; or, you can assign a custom icon using the **MousePointer** and **MouseIcon** properties.

This example works in the following ways:

- Choose a mouse pointer from the **ListBox** to change the mouse pointer associated with the first **CommandButton**.
- Click the first **CommandButton** to associate its mouse pointer with the second **CommandButton**.
- Click the second **CommandButton** to load a custom icon for its mouse pointer.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- Two **CommandButton** controls named CommandButton1 and CommandButton2.
- A **ListBox** named ListBox1.

Note This example uses two icon files (identified by the ico file extention) that are loaded using the **LoadPicture** function. You should edit each **LoadPicture** function call to specify an icon file that resides on your system.

```
Private Sub ListBox1_Click()
    If IsNull(ListBox1.Value) = False Then
        CommandButton1.MousePointer = ListBox1.Value

        If CommandButton1.MousePointer = fmMousePointerCustom Then
            CommandButton1.MouseIcon = LoadPicture("c:\msvc20\cdk32\
samples\circ1\bix.ico")
        End If
    End If
End Sub

Private Sub CommandButton1_Click()
    CommandButton2.MousePointer = CommandButton1.MousePointer

    If CommandButton2.MousePointer = fmMousePointerCustom Then
        CommandButton2.MouseIcon = CommandButton1.MouseIcon
    End If
End Sub

Private Sub CommandButton2_Click()
    CommandButton2.MousePointer = fmMousePointerCustom
    CommandButton2.MouseIcon = LoadPicture("c:\msvc20\cdk32\samples\push\
push.ico")
End Sub

Private Sub UserForm_Initialize()
    'Load ListBox with MousePointer choices
    ListBox1.ColumnCount = 2

    ListBox1.AddItem "fmMousePointerDefault"
    ListBox1.List(0, 1) = fmMousePointerDefault
```

```
    ListBox1.AddItem "fmMousePointerArrow"
    ListBox1.List(1, 1) = fmMousePointerArrow
    ListBox1.AddItem "fmMousePointerCross"
    ListBox1.List(2, 1) = fmMousePointerCross

    ListBox1.AddItem "fmMousePointerIBeam"
    ListBox1.List(3, 1) = fmMousePointerIBeam
    ListBox1.AddItem "fmMousePointerSizeNESW"
    ListBox1.List(4, 1) = fmMousePointerSizeNESW
    ListBox1.AddItem "fmMousePointerSizeNS"
    ListBox1.List(5, 1) = fmMousePointerSizeNS

    ListBox1.AddItem "fmMousePointerSizeNWSE"
    ListBox1.List(6, 1) = fmMousePointerSizeNWSE
    ListBox1.AddItem "fmMousePointerSizeWE"
    ListBox1.List(7, 1) = fmMousePointerSizeWE
    ListBox1.AddItem "fmMousePointerUpArrow"
    ListBox1.List(8, 1) = fmMousePointerUpArrow

    ListBox1.AddItem "fmMousePointerHourglass"
    ListBox1.List(9, 1) = fmMousePointerHourGlass
    ListBox1.AddItem "fmMousePointerNoDrop"
    ListBox1.List(10, 1) = fmMousePointerNoDrop
    ListBox1.AddItem "fmMousePointerAppStarting"
    ListBox1.List(11, 1) = fmMousePointerAppStarting

    ListBox1.AddItem "fmMousePointerHelp"
    ListBox1.List(12, 1) = fmMousePointerHelp
    ListBox1.AddItem "fmMousePointerSizeAll"
    ListBox1.List(13, 1) = fmMousePointerSizeAll
    ListBox1.AddItem "fmMousePointerCustom"
    ListBox1.List(14, 1) = fmMousePointerCustom

    ListBox1.BoundColumn = 2
    ListBox1.Value = fmMousePointerDefault

    MsgBox "ListBox1.Value =" & ListBox1.Value & "."
    CommandButton1.MousePointer = ListBox1.Value
End Sub
```

EnterKeyBehavior Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpEnterKeyBehaviorC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpEnterKeyBehaviorA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpEnterKeyBehaviorS"}

The following example uses the **EnterKeyBehavior** property to control the effect of ENTER in a **TextBox**. In this example, the user can specify either a single-line or multiline **TextBox**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **TextBox** named TextBox1.
- Two **ToggleButton** controls named ToggleButton1 and ToggleButton2.

```
Private Sub UserForm_Initialize()
    TextBox1.EnterKeyBehavior = True
    ToggleButton1.Caption = "EnterKeyBehavior is True"
    ToggleButton1.Width = 70
    ToggleButton1.Value = True

    TextBox1.MultiLine = True
    ToggleButton2.Caption = "MultiLine is True"
    ToggleButton2.Width = 70
    ToggleButton2.Value = True

    TextBox1.Height = 100
    TextBox1.WordWrap = True
    TextBox1.Text = "Type your text here. If EnterKeyBehavior is True,"& _
    " press Enter to start a new line. Otherwise, press SHIFT+ENTER."
End Sub

Private Sub ToggleButton1_Click()
    If ToggleButton1.Value = True Then
        TextBox1.EnterKeyBehavior = True
        ToggleButton1.Caption = "EnterKeyBehavior is True"
    Else
        TextBox1.EnterKeyBehavior = False
        ToggleButton1.Caption = "EnterKeyBehavior is False"
    End If
End Sub

Private Sub ToggleButton2_Click()
    If ToggleButton2.Value = True Then
        TextBox1.MultiLine = True
        ToggleButton2.Caption = "MultiLine TextBox"
    Else
        TextBox1.MultiLine = False
        ToggleButton2.Caption = "Single-line TextBox"
    End If
End Sub
```

Index Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpIndexC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpIndexA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpIndexS"}

The following example uses the **Index** property to change the order of the pages and tabs in a **MultiPage** and **TabStrip**. The user chooses CommandButton1 to move the third page and tab to the front of the **MultiPage** and **TabStrip.** The user chooses CommandButton2 to move the selected page and tab to the back of the **MultiPage** and **TabStrip**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- Two **CommandButton** controls named CommandButton1 and CommandButton2.
- A **MultiPage** named MultiPage1.
- A **TabStrip** named TabStrip1.

```
Dim MyPageOrTab As Object

Private Sub CommandButton1_Click()
'Move third page and tab to front of control
    MultiPage1.page3.Index = 0
    TabStrip1.Tab3.Index = 0
End Sub

Private Sub CommandButton2_Click()
'Move selected page and tab to back of control
    Set MyPageOrObject = MultiPage1.SelectedItem
    MsgBox "MultiPage1.SelectedItem = " & MultiPage1.SelectedItem.Name
    MyPageOrObject.Index = 4

    Set MyPageOrObject = TabStrip1.SelectedItem
    MsgBox "TabStrip1.SelectedItem = " & TabStrip1.SelectedItem.Caption
    MyPageOrObject.Index = 4
End Sub

Private Sub UserForm_Initialize()
    MultiPage1.Width = 200
    MultiPage1.Pages.Add
    MultiPage1.Pages.Add
    MultiPage1.Pages.Add

    TabStrip1.Width = 200
    TabStrip1.Tabs.Add
    TabStrip1.Tabs.Add
    TabStrip1.Tabs.Add

    CommandButton1.Caption = "Move third page/tab to front"
    CommandButton1.Width = 120

    CommandButton2.Caption = "Move selected item to back"
    CommandButton2.Width = 120
 End Sub
```

Enabled, Locked Properties Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpEnabledLockedC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpEnabledLockedA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpEnabledLockedS"}

The following example demonstrates the **Enabled** and **Locked** properties and how they complement each other. This example exposes each property independently with a **CheckBox**, so you observe the settings individually and combined. This example also includes a second **TextBox** so you can copy and paste information between the **TextBox** controls and verify the activities supported by the settings of these properties.

Note You can copy the selection to the Clipboard using CTRL+C and paste using CTRL+V.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **TextBox** named TextBox1.
- Two **CheckBox** controls named CheckBox1 and CheckBox2.
- A second **TextBox** named TextBox2.

```
Private Sub CheckBox1_Change()
    TextBox2.Text = "TextBox2"
    TextBox1.Enabled = CheckBox1.Value
End Sub

Private Sub CheckBox2_Change()
    TextBox2.Text = "TextBox2"
    TextBox1.Locked = CheckBox2.Value
End Sub

Private Sub UserForm_Initialize()
    TextBox1.Text = "TextBox1"
    TextBox1.Enabled = True
    TextBox1.Locked = False

    CheckBox1.Caption = "Enabled"
    CheckBox1.Value = True

    CheckBox2.Caption = "Locked"
    CheckBox2.Value = False

    TextBox2.Text = "TextBox2"
End Sub
```

Delay Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpDelayC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpDelayA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpDelayS"}

The following example demonstrates the time interval between successive Change, SpinUp, and SpinDown events that occur when a user holds down the mouse button to change the value of a **SpinButton** or **ScrollBar**.

In this example, the user chooses a delay setting, then clicks and holds down either side of a **SpinButton**. The SpinUp and SpinDown events are recorded in a **ListBox** as they are initiated.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **SpinButton** named SpinButton1.
- Two **OptionButton** controls named OptionButton1 and OptionButton2.
- A **ListBox** named ListBox1.

```
Dim EventCount As Long

Private Sub ResetControl()
    ListBox1.Clear
    EventCount = 0
    SpinButton1.Value = 5000
End Sub

Private Sub UserForm_Initialize()
    SpinButton1.Min = 0
    SpinButton1.Max = 10000
    ResetControl

    SpinButton1.Delay = 50
    OptionButton1.Caption = "50 millisecond delay"
    OptionButton2.Caption = "250 millisecond delay"

    OptionButton1.Value = True
End Sub

Private Sub OptionButton1_Click()
    SpinButton1.Delay = 50
    ResetControl
End Sub

Private Sub OptionButton2_Click()
    SpinButton1.Delay = 250
    ResetControl
End Sub

Private Sub SpinButton1_SpinDown()
    EventCount = EventCount + 1
    ListBox1.AddItem EventCount
End Sub

Private Sub SpinButton1_SpinUp()
    EventCount = EventCount + 1
    ListBox1.AddItem EventCount
End Sub
```

DropButtonStyle, ShowDropButtonWhen Properties Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpDropButtonStyleShowDropButtonWhenC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpDropButtonStyleShowDropButtonWhenA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpDropButtonStyleShowDropButtonWhenS"}

The following example demonstrates the different symbols that you can specify for a drop-down arrow in a **ComboBox** or **TextBox**. In this example, the user chooses a drop-down arrow style from a **ComboBox**. This example also uses the **Locked** property. To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **ComboBox** named ComboBox1.
- A **Label** named Label1.
- A **TextBox** named TextBox1 placed beneath Label1.

```
Private Sub ComboBox1_Click()
    ComboBox1.DropButtonStyle = ComboBox1.Value
    TextBox1.DropButtonStyle = ComboBox1.Value
End Sub

Private Sub UserForm_Initialize()
    ComboBox1.ColumnCount = 2
    ComboBox1.BoundColumn = 2
    ComboBox1.TextColumn = 1

    ComboBox1.AddItem "Blank Button"
    ComboBox1.List(0, 1) = 0
    ComboBox1.AddItem "Down Arrow"
    ComboBox1.List(1, 1) = 1
    ComboBox1.AddItem "Ellipsis"
    ComboBox1.List(2, 1) = 2
    ComboBox1.AddItem "Underscore"
    ComboBox1.List(3, 1) = 3

    ComboBox1.Value = 0

    TextBox1.Text = "TextBox1"
    TextBox1.ShowDropButtonWhen = fmShowDropButtonWhenAlways
    TextBox1.Locked = True

    Label1.Caption = "TheDropButton also applies to a TextBox."
    Label1.AutoSize = True
    Label1.WordWrap = False
End Sub
```

LineCount, TextLength Properties Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpLineCountTextLengthC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpLineCountTextLengthA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpLineCountTextLengthS"}

The following example counts the characters and the number of lines of text in a **TextBox** by using the **LineCount** and **TextLength** properties, and the **SetFocus** method. In this example, the user can type into a **TextBox**, and can retrieve current values of the **LineCount** and **TextLength** properties.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains the following controls:

- A **TextBox** named TextBox1.
- A **CommandButton** named CommandButton1.
- Two **Label** controls named Label1 and Label2.

```
'Type SHIFT+ENTER to start a new line in the text box.

Private Sub CommandButton1_Click()
    'Must first give TextBox1 the focus to get line count
    TextBox1.SetFocus
    Label1.Caption = "LineCount = " & TextBox1.LineCount
    Label2.Caption = "TextLength = " & TextBox1.TextLength
End Sub

Private Sub UserForm_Initialize()
    CommandButton1.WordWrap = True
    CommandButton1.AutoSize = True
    CommandButton1.Caption = "Get Counts"

    Label1.Caption = "LineCount = "
    Label2.Caption = "TextLength = "

    TextBox1.MultiLine = True
    TextBox1.WordWrap = True
    TextBox1.Text = "Enter your text here."
End Sub
```

Count Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpCountC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpCountA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpCountS"}

The following example displays the **Count** property of the **Controls** collection for the form, and the **Count** property identifying the number of pages and tabs of each **MultiPage** and **TabStrip**.

To use this example, copy this sample code to the Declarations portion of a form. The form can contain any number of controls, with the following restrictions:

- Names of **MultiPage** controls must start with “MultiPage”.
- Names of **TabStrip** controls must start with “TabStrip”.

Note You can add pages to a **MultiPage** or add tabs to a **TabStrip**. Double-click on the control, then right click in the tab area of the control and choose **New Page** from the shortcut menu.

```
Private Sub UserForm_Initialize()
    Dim MyControl As Control

    MsgBox "UserForm1.Controls.Count = " & Controls.Count

    For Each MyControl In Controls
        If (MyControl.Name Like "MultiPage*") Then
            MsgBox MyControl.Name & ".Pages.Count = " &
MyControl.Pages.Count
        ElseIf (MyControl.Name Like "TabStrip*") Then
            MsgBox MyControl.Name & ".Tabs.Count = " & MyControl.Tabs.Count
        End If
    Next

End Sub
```

Alignment Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpAlignmentC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpAlignmentA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpAlignmentS"}

The following example demonstrates the **Alignment** property used with several **OptionButton** controls. In this example, the user can change the alignment by clicking a **ToggleButton**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains the following controls:

- Two **OptionButton** controls named OptionButton1 and OptionButton2.
- A **ToggleButton** named ToggleButton1.

```
Private Sub UserForm_Initialize()
    OptionButton1.Alignment = fmAlignmentLeft
    OptionButton2.Alignment = fmAlignmentLeft

    OptionButton1.Caption = "Alignment with AutoSize"
    OptionButton2.Caption = "Choice 2"
    OptionButton1.AutoSize = True
    OptionButton2.AutoSize = True

    ToggleButton1.Caption = "Left Align"
    ToggleButton1.WordWrap = True
    ToggleButton1.Value = True
End Sub

Private Sub ToggleButton1_Click()
    If ToggleButton1.Value = True Then
        ToggleButton1.Caption = "Left Align"
        OptionButton1.Alignment = fmAlignmentLeft
        OptionButton2.Alignment = fmAlignmentLeft
    Else
        ToggleButton1.Caption = "Right Align"
        OptionButton1.Alignment = fmAlignmentRight
        OptionButton2.Alignment = fmAlignmentRight
    End If
End Sub
```

ActiveControl Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpActiveControlC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpActiveControlA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpActiveControlS"}

The following example uses the **ActiveControl** property in a subroutine that tracks the controls a user visits. The Enter event for each control calls the TraceFocus subroutine to identify the control that has the focus.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains the following controls:

- A **ScrollBar** named ScrollBar1.
- A **ListBox** named ListBox1.
- Two **OptionButton** controls named OptionButton1 and OptionButton2.
- A **Frame** named Frame1.

```
Dim MyControl As Control

Private Sub TraceFocus()
    ListBox1.AddItem ActiveControl.Name
    ListBox1.List(ListBox1.ListCount - 1, 1) = ActiveControl.TabIndex
End Sub

Private Sub UserForm_Initialize()
    ListBox1.ColumnCount = 2
    ListBox1.AddItem "Controls Visited"
    ListBox1.List(0, 1) = "Control Index"
End Sub

Private Sub Frame1_Enter()
    TraceFocus
End Sub

Private Sub ListBox1_Enter()
    TraceFocus
End Sub

Private Sub OptionButton1_Enter()
    TraceFocus
End Sub

Private Sub OptionButton2_Enter()
    TraceFocus
End Sub

Private Sub ScrollBar1_Enter()
    TraceFocus
End Sub
```

DropDown Method Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpDropDownC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpDropDownA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpDropDownS"}

The following example uses the **DropDown** method to display the list in a **ComboBox**. The user can display the list of a **ComboBox** by clicking the **CommandButton**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **ComboBox** named ComboBox1.
- A **CommandButton** named CommandButton1.

```
Private Sub CommandButton1_Click()
    ComboBox1.DropDown
End Sub

Private Sub UserForm_Initialize()
    ComboBox1.AddItem "Turkey"
    ComboBox1.AddItem "Chicken"
    ComboBox1.AddItem "Duck"
    ComboBox1.AddItem "Goose"
    ComboBox1.AddItem "Grouse"
End Sub
```

Cut and Paste From a TextBox Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpCutPasteFromTextBoxC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpCutPasteFromTextBoxA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpCutPasteFromTextBoxS"}

The following example uses the **Cut** and **Paste** methods to cut text from one **TextBox** and paste it into another **TextBox**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- Two **TextBox** controls named TextBox1 and TextBox2.
- A **CommandButton** named CommandButton1.

```
Private Sub UserForm_Initialize()
    TextBox1.Text = "From TextBox1!"
    TextBox2.Text = "Hello "

    CommandButton1.Caption = "Cut and Paste"
    CommandButton1.AutoSize = True
End Sub

Private Sub CommandButton1_Click()
    TextBox2.SelStart = 0
    TextBox2.SelLength = TextBox2.TextLength
    TextBox2.Cut

    TextBox1.SetFocus
    TextBox1.SelStart = 0

    TextBox1.Paste
    TextBox2.SelStart = 0
End Sub
```

Cut and Paste From a Page Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpCutPasteFromPageC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpCutPasteFromPageA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpCutPasteFromPageS"}

The following example uses the **Add**, **Cut**, and **Paste** methods to cut and paste a control from a **Page** of a **MultiPage**. The control involved in the cut and paste operations is dynamically added to the form.

This example assumes the user will add, then cut, then paste the new control.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- Three **CommandButton** controls named CommandButton1 through CommandButton3.
- A **MultiPage** named MultiPage1.

```
Dim MyTextBox As Control

Private Sub CommandButton1_Click()
    Set MyTextBox =
MultiPage1.Pages(MultiPage1.Value).Controls.Add("Forms.TextBox.1",
"MyTextBox", Visible)
    CommandButton2.Enabled = True
    CommandButton1.Enabled = False
End Sub
Private Sub CommandButton2_Click()
    MultiPage1.Pages(MultiPage1.Value).Controls.Cut
    CommandButton3.Enabled = True
    CommandButton2.Enabled = False
End Sub
Private Sub CommandButton3_Click()
    Dim MyPage As Object
    Set MyPage = MultiPage1.Pages.Item(MultiPage1.Value)

    MyPage.Paste
    CommandButton3.Enabled = False
End Sub
Private Sub UserForm_Initialize()
    CommandButton1.Caption = "Add"
    CommandButton2.Caption = "Cut"
    CommandButton3.Caption = "Paste"

    CommandButton1.Enabled = True
    CommandButton2.Enabled = False
    CommandButton3.Enabled = False
End Sub
```

Name Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpNameC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpNameA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpNameS"}

The following example displays the **Name** property of each control on a form. This example uses the **Controls** collection to cycle through all the controls placed directly on the Userform.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains a **CommandButton** named CommandButton1 and several other controls.

```
Private Sub CommandButton1_Click()
    Dim MyControl As Control

    For Each MyControl In Controls
        MsgBox "MyControl.Name = " & MyControl.Name
    Next
End Sub
```

Accessing a Tab Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpAccessingTabC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpAccessingTabA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpAccessingTabS"}

The following example accesses an individual tab of a **TabStrip** in several ways:

- Using the **Tabs** collection with a numeric index.
- Using the **Tabs** collection with a string index.
- Using the **Tabs** collection with the **Item** method.
- Using the name of the individual **Tab**.
- Using the **SelectedItem** property.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains a **TabStrip** named TabStrip1.

```
Private Sub UserForm_Initialize()
    Dim TabName As String

    For i = 0 To TabStrip1.Count - 1
        'Using index (numeric or string)
        MsgBox "TabStrip1.Tabs(i).Caption = " & TabStrip1.Tabs(i).Caption
        MsgBox "TabStrip1.Tabs.Item(i).Caption = " &
TabStrip1.Tabs.Item(i).Caption

        TabName = TabStrip1.Tabs(i).Name
        MsgBox "TabName = " & TabName

        MsgBox "TabStrip1.Tabs(TabName).Caption = " &
TabStrip1.Tabs(TabName).Caption
        MsgBox "TabStrip1.Tabs.Item(TabName).Caption = " &
TabStrip1.Tabs.Item(TabName).Caption

        'Use Tab object without referring to Tabs collection
        If i = 0 Then
            MsgBox "TabStrip1.Tab1. Caption = " & TabStrip1.Tab1.Caption
        ElseIf i = 1 Then
            MsgBox "TabStrip1.Tab2. Caption = " & TabStrip1.Tab2.Caption
        EndIf

        'Use SelectedItem Property
        TabStrip1.Value = i
        MsgBox " TabStrip1.SelectedItem.Caption = " &
TabStrip1.SelectedItem.Caption
    Next i
End Sub
```

Zoom Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpZoomPropC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpZoomPropA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpZoomPropS"}

The following example uses the **Zoom** property to shrink or enlarge the information displayed on a form, Page, or Frame. This example includes a **Frame**, a **TextBox** in the **Frame**, and a **ScrollBar**. The magnification level of the **Frame** changes through **Zoom**. The user can set **Zoom** by using the **ScrollBar**. The **TextBox** is present to demonstrate the effects of zooming.

This example also uses the **Max** and **Min** properties to identify the range of acceptable values for the **ScrollBar**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **Label** named Label1.
- A **ScrollBar** named ScrollBar1.
- A second **Label** named Label2.
- A **Frame** named Frame1.
- A **TextBox** named TextBox1 that is located inside Frame1.

```
Private Sub UserForm_Initialize()
    ScrollBar1.Max = 400
    ScrollBar1.Min = 10
    ScrollBar1.Value = 100

    Label1.Caption = "10  -----Percent of Original Size---- 400"
    Label2.Caption = ScrollBar1.Value

    Frame1.TextBox1.Text = "Enter your text here."
    Frame1.TextBox1.MultiLine = True
    Frame1.TextBox1.WordWrap = True

    Frame1.Zoom = ScrollBar1.Value
End Sub

Private Sub ScrollBar1_Change()
    Frame1.Zoom = ScrollBar1.Value
    Label2.Caption = ScrollBar1.Value
End Sub
```

TextColumn Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpTextColumnC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpTextColumnA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpTextColumnS"}

The following example uses the **TextColumn** property to identify the column of data in a **ListBox** that supplies data for its **Text** property. This example sets the third column of the **ListBox** as the text column. As you select an entry from the **ListBox**, the value from the **TextColumn** will be displayed in the **Label**.

This example also demonstrates how to load a multicolumn **ListBox** using the **AddItem** method and the **List** property.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **ListBox** named ListBox1.
- A **TextBox** named TextBox1.

```
Private Sub UserForm_Initialize()
ListBox1.ColumnCount = 3

ListBox1.AddItem "Row 1, Col 1"
ListBox1.List(0, 1) = "Row 1, Col 2"
ListBox1.List(0, 2) = "Row 1, Col 3"

ListBox1.AddItem "Row 2, Col 1"
ListBox1.List(1, 1) = "Row 2, Col 2"
ListBox1.List(1, 2) = "Row 2, Col 3"

ListBox1.AddItem "Row 3, Col 1"
ListBox1.List(2, 1) = "Row 3, Col 2"
ListBox1.List(2, 2) = "Row 3, Col 3"

ListBox1.TextColumn = 3
End Sub

Private Sub ListBox1_Change()
TextBox1.Text = ListBox1.Text
End Sub
```

PictureSizeMode Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpPictureSizeModePropertyC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpPictureSizeModePropertyA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpPictureSizeModePropertyS"}

The following example uses the **PictureSizeMode** property to demonstrate three display options for a picture: showing the picture as is, changing the size of the picture while maintaining its original proportions, and stretching the picture to fill a space.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **Frame** named Frame1.
- A **SpinButton** named SpinButton1.
- A **TextBox** named TextBox1.
- Three **OptionButton** controls named OptionButton1 through OptionButton3.

Note This example is an enhanced version of the **PictureAlignment** property example, as the two properties complement each other. The enhancements are three **OptionButton** event subroutines that control whether the image is cropped, zoomed, or stretched.

```
Dim Alignments(5) As String

Private Sub UserForm_Initialize()
    Alignments(0) = "0 - Top Left"
    Alignments(1) = "1 - Top Right"
    Alignments(2) = "2 - Center"
    Alignments(3) = "3 - Bottom Left"
    Alignments(4) = "4 - Bottom Right"

    'Specify a bitmap that exists on your system
    Frame1.Picture = LoadPicture("c:\winnt2\ball.bmp")

    SpinButton1.Min = 0
    SpinButton1.Max = 4
    SpinButton1.Value = 0

    TextBox1.Text = Alignments(0)
    Frame1.PictureAlignment = SpinButton1.Value

    OptionButton1.Caption = "Crop"
    OptionButton1.Value = True
    OptionButton2.Caption = "Stretch"
    OptionButton3.Caption = "Zoom"
End Sub

Private Sub OptionButton1_Click()
    If OptionButton1.Value = True Then
        Frame1.PictureSizeMode = fmPictureSizeModeClip
    End If
End Sub

Private Sub OptionButton2_Click()
    If OptionButton2.Value = True Then
        Frame1.PictureSizeMode = fmPictureSizeModeStretch
    End If
End Sub
```

```
Private Sub OptionButton3_Click()
    If OptionButton3.Value = True Then
        Frame1.PictureSizeMode = fmPictureSizeModeZoom
    End If
End Sub

Private Sub SpinButton1_Change()
    TextBox1.Text = Alignments(SpinButton1.Value)
    Frame1.PictureAlignment = SpinButton1.Value
End Sub

Private Sub TextBox1_Change()
    Select Case TextBox1.Text
    Case "0"
        TextBox1.Text = Alignments(0)
        Frame1.PictureAlignment = 0
    Case "1"
        TextBox1.Text = Alignments(1)
        Frame1.PictureAlignment = 1
    Case "2"
        TextBox1.Text = Alignments(2)
        Frame1.PictureAlignment = 2
    Case "3"
        TextBox1.Text = Alignments(3)
        Frame1.PictureAlignment = 3
    Case "4"
        TextBox1.Text = Alignments(4)
        Frame1.PictureAlignment = 4
    Case Else
        TextBox1.Text = Alignments(SpinButton1.Value)
        Frame1.PictureAlignment = SpinButton1.Value
    End Select
End Sub
```

PictureAlignment Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpPictureAlignmentC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpPictureAlignmentA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpPictureAlignmentS"}

The following example uses the **PictureAlignment** property to set up a background picture. The example also identifies the alignment options provided by **PictureAlignment**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **Frame** named Frame1.
- A **SpinButton** named SpinButton1.
- A **TextBox** named TextBox1.

```
Dim Alignments(5) As String

Private Sub UserForm_Initialize()
    Alignments(0) = "0 - Top Left"
    Alignments(1) = "1 - Top Right"
    Alignments(2) = "2 - Center"
    Alignments(3) = "3 - Bottom Left"
    Alignments(4) = "4 - Bottom Right"

    'Specify a bitmap that exists on your system
    Frame1.Picture = LoadPicture("c:\winnt2\ball.bmp")

    SpinButton1.Min = 0
    SpinButton1.Max = 4
    SpinButton1.Value = 0

    TextBox1.Text = Alignments(0)
    Frame1.PictureAlignment = SpinButton1.Value
End Sub

Private Sub SpinButton1_Change()
    TextBox1.Text = Alignments(SpinButton1.Value)
    Frame1.PictureAlignment = SpinButton1.Value
End Sub

Private Sub TextBox1_Change()
    Select Case TextBox1.Text
    Case "0"
        TextBox1.Text = Alignments(0)
        Frame1.PictureAlignment = 0
    Case "1"
        TextBox1.Text = Alignments(1)
        Frame1.PictureAlignment = 1
    Case "2"
        TextBox1.Text = Alignments(2)
        Frame1.PictureAlignment = 2
    Case "3"
        TextBox1.Text = Alignments(3)
        Frame1.PictureAlignment = 3
    Case "4"
        TextBox1.Text = Alignments(4)
        Frame1.PictureAlignment = 4
    Case Else
```

```
        TextBox1.Text = Alignments(SpinButton1.Value)
        Frame1.PictureAlignment = SpinButton1.Value
    End Select
End Sub
```

GroupName Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpGroupNameC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpGroupNameA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpGroupNameS"}

The following example uses the **GroupName** property to create two groups of **OptionButton** controls on the same form.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains five **OptionButton** controls named OptionButton1 through OptionButton5.

```
Private Sub UserForm_Initialize()
    OptionButton1.GroupName = "Widgets"
    OptionButton2.GroupName = "Widgets"
    OptionButton4.GroupName = "Widgets"

    OptionButton3.GroupName = "Gadgets-Group2"
    OptionButton5.GroupName = "Gadgets-Group2"
End Sub
```

GetFromClipboard Method Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpGetFromClipboardC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpGetFromClipboardA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpGetFromClipboardS"}

The following example demonstrates data movement from a **TextBox** to the Clipboard, from the Clipboard to a **DataObject**, and from a **DataObject** into another **TextBox**. The **GetFromClipboard** method transfers the data from the Clipboard to a **DataObject**. The **Copy** and **GetText** methods are also used.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- Two **TextBox** controls named TextBox1 and TextBox2.
- A **CommandButton** named CommandButton1.

```
Dim MyData as DataObject

Private Sub CommandButton1_Click()
    'Need to select text before copying it to Clipboard
    TextBox1.SelStart = 0
    TextBox1.SelLength = TextBox1.TextLength
    TextBox1.Copy

    MyData.GetFromClipboard
    TextBox2.Text = MyData.GetText(1)
End Sub

Private Sub UserForm_Initialize()
    Set MyData = New DataObject
    TextBox1.Text = "Move this data to the Clipboard, to a DataObject, then
to TextBox2!"
End Sub
```

PutInClipboard Method Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpPutInClipboardC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpPutInClipboardA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpPutInClipboardS"}

The following example demonstrates data movement from a **TextBox** to a **DataObject**, from a **DataObject** to the Clipboard, and from the Clipboard to another **TextBox**. The **PutInClipboard** method transfers the data from a **DataObject** to the Clipboard. The **SetText** and **Paste** methods are also used.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- Two **TextBox** controls named TextBox1 and TextBox2.
- A **CommandButton** named CommandButton1.

```
Dim MyData As DataObject

Private Sub CommandButton1_Click()
    Set MyData = New DataObject

    MyData.SetText TextBox1.Text
    MyData.PutInClipboard

    TextBox2.Paste
End Sub

Private Sub UserForm_Initialize()
    TextBox1.Text = "Move this data to a DataObject, to the Clipboard, then
to TextBox2!"
End Sub
```

DragBehavior, EnterFieldBehavior Properties Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpDragBehaviorEnterFieldBehaviorC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpDragBehaviorEnterFieldBehaviorA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpDragBehaviorEnterFieldBehaviorS"}

The following example uses the **DragBehavior** and **EnterFieldBehavior** properties to demonstrate the different effects that you can provide when entering a control and when dragging information from one control to another.

The sample uses two **TextBox** controls. You can set **DragBehavior** and **EnterFieldBehavior** for each control and see the effects of dragging from one control to another.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **TextBox** named TextBox1.
- Two **ToggleButton** controls named ToggleButton1 and ToggleButton2. These controls are associated with TextBox1.
- A **TextBox** named TextBox2.
- Two **ToggleButton** controls named ToggleButton3 and ToggleButton4. These controls are associated with TextBox2.

```
Private Sub UserForm_Initialize()
    TextBox1.Text = "Once upon a time in a land ...,"
    ToggleButton1.Value = True
    ToggleButton1.Caption = "Drag Enabled"
    ToggleButton1.WordWrap = True
    TextBox1.DragBehavior = fmDragBehaviorEnabled

    ToggleButton2.Value = True
    ToggleButton2.Caption = "Recall Selection"
    ToggleButton2.WordWrap = True
    TextBox1.EnterFieldBehavior = fmEnterFieldBehaviorRecallSelection

    TextBox2.Text = "XXX, YYYY"
    ToggleButton3.Value = False
    ToggleButton3.Caption = "Drag Disabled"
    ToggleButton3.WordWrap = True
    TextBox2.DragBehavior = fmDragBehaviorDisabled

    ToggleButton4.Value = False
    ToggleButton4.Caption = "Select All"
    ToggleButton4.WordWrap = True
    TextBox2.EnterFieldBehavior = fmEnterFieldBehaviorSelectAll
End Sub

Private Sub ToggleButton1_Click()
    If ToggleButton1.Value = True Then
       ToggleButton1.Caption = "Drag Enabled"
       TextBox1.DragBehavior = fmDragBehaviorEnabled
    Else
       ToggleButton1.Caption = "Drag Disabled"
       TextBox1.DragBehavior = fmDragBehaviorDisabled
    End If
End Sub

Private Sub ToggleButton2_Click()
```

```
    If ToggleButton2.Value = True Then
        ToggleButton2.Caption = "Recall Selection"
        TextBox1.EnterFieldBehavior = fmEnterFieldBehaviorRecallSelection
    Else
        ToggleButton2.Caption = "Select All"
        TextBox1.EnterFieldBehavior = fmEnterFieldBehaviorSelectAll
    End If
End Sub

Private Sub ToggleButton3_Click()
    If ToggleButton3.Value = True Then
       ToggleButton3.Caption = "Drag Enabled"
       TextBox2.DragBehavior = fmDragBehaviorEnabled
    Else
       ToggleButton3.Caption = "Drag Disabled"
       TextBox2.DragBehavior = fmDragBehaviorDisabled
    End If
End Sub

Private Sub ToggleButton4_Click()
    If ToggleButton4.Value = True Then
        ToggleButton4.Caption = "Recall Selection"
        TextBox2.EnterFieldBehavior = fmEnterFieldBehaviorRecallSelection
    Else
       ToggleButton4.Caption = "Select All"
       TextBox2.EnterFieldBehavior = fmEnterFieldBehaviorSelectAll
    End If
End Sub
```

GetFormat, GetText, SetText Methods Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpGetFormatGetTextSetTextC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpGetFormatGetTextSetTextA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpGetFormatGetTextSetTextS"}

The following example uses the **GetFormat**, **GetText**, and **SetText** methods to transfer text between a **DataObject** and the Clipboard.

The user types text into a **TextBox** and then can transfer it to a **DataObject** in a standard text format by clicking CommandButton1. Clicking CommandButton2 retrieves the text from the **DataObject**. Clicking CommandButton3 copies text from TextBox1 to the **DataObject** in a custom format. Clicking CommandButton4 retrieves the text from the **DataObject** in a custom format.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **TextBox** named TextBox1.
- Four **CommandButton** controls named CommandButton1 through CommandButton4.
- A **Label** named Label1.

```
Dim MyDataObject As DataObject

Private Sub CommandButton1_Click()
'Put standard format on Clipboard
    If TextBox1.TextLength > 0 Then
        Set MyDataObject = New DataObject
        MyDataObject.SetText TextBox1.Text
        Label1.Caption = "Put on D.O."
        CommandButton2.Enabled = True
        CommandButton4.Enabled = False
    End If
End Sub

Private Sub CommandButton2_Click()
'Get standard format from Clipboard
    If MyDataObject.GetFormat(1) = True Then
        Label1.Caption = "Std format - " & MyDataObject.GetText(1)
    End If
End Sub

Private Sub CommandButton3_Click()
'Put custom format on Clipboard
    If TextBox1.TextLength > 0 Then
        Set MyDataObject = New DataObject
        MyDataObject.SetText TextBox1.Text, 233
        Label1.Caption = "Custom on D.O."
        CommandButton4.Enabled = True
        CommandButton2.Enabled = False
    End If
End Sub

Private Sub CommandButton4_Click()
'Get custom format from Clipboard
    If MyDataObject.GetFormat(233) = True Then
        Label1.Caption = "Cust format - " & MyDataObject.GetText(233)
End If
End Sub
```

```
Private Sub UserForm_Initialize()
    CommandButton2.Enabled = False
    CommandButton4.Enabled = False
End Sub
```

CanPaste Property, Paste, Copy Methods Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpCanPasteCopyC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpCanPasteCopyA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpCanPasteCopyS"}

The following example uses the **CanPaste** property and the **Paste** method to paste a **ComboBox** from the Clipboard to a **Page** of a **MultiPage**. This sample also uses the **SetFocus** and **Copy** methods to copy a control from the form to the Clipboard.

The user clicks CommandButton1 to copy the **ComboBox** to the Clipboard. The user double-clicks (using the DblClick event) CommandButton1 to paste the **ComboBox** to the **MultiPage**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **TextBox** named TextBox1.
- A **ComboBox** named ComboBox1.
- A **MultiPage** named MultiPage1.
- A **CommandButton** named CommandButton1.

Note This example also includes a subroutine to illustrate pasting text into a control.

```
Private Sub UserForm_Initialize()
    ComboBox1.AddItem "It's a beautiful day!"

    CommandButton1.Caption = "Copy ComboBox to Clipboard"
    CommandButton1.AutoSize = True
End Sub

Private Sub MultiPage1_DblClick(ByVal Index As Long, ByVal Cancel As
MSForms.ReturnBoolean)
    If MultiPage1.Pages(MultiPage1.Value).CanPaste = True Then
        MultiPage1.Pages(MultiPage1.Value).Paste
    Else
        TextBox1.Text = "Can't Paste"
    End If
End Sub

Private Sub CommandButton1_Click()
        UserForm1.ComboBox1.SetFocus
        UserForm1.Copy
End Sub

'Code for pasting text into a control
'Private Sub ComboBox1_DblClick(ByVal Cancel As MSForms.ReturnBoolean)
'   If ComboBox1.CanPaste = True Then
'        ComboBox1.Paste
'    Else
'        TextBox1.Text = "Can't Paste"
'    End If
   'End Sub
```

ScrollBar Control Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpScrollBarC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpScrollBarA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpScrollBarS"}

The following example demonstrates the stand-alone **ScrollBar** and reports the change in its value as the user moves the scroll box. The user can move the scroll box by clicking on either arrow at the ends of the control, by clicking in the region between scroll box and either arrow, or by dragging the scroll box. When the user drags the scroll box, the Scroll event displays a message indicating that the user scrolled to obtain the new value.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **ScrollBar** named ScrollBar1.
- Two **Label** controls named Label1 and Label2. Label1 contains scaling information for the user. Label2 reports the delta value.

```
Dim ScrollSaved As Integer        'Previous ScrollBar setting

Private Sub UserForm_Initialize()
    ScrollBar1.Min = -225
    ScrollBar1.Max = 289
    ScrollBar1.Value = 0

    Label1.Caption = "-225  -----Widgets-----   289"
    Label1.AutoSize = True

    Label2.Caption = ""
End Sub

Private Sub ScrollBar1_Change()
    Label2.Caption = " Widget Changes " & (ScrollSaved - ScrollBar1.Value)
End Sub

Private Sub ScrollBar1_Exit(ByVal Cancel as MSForms.ReturnBoolean)
    Label2.Caption = " Widget Changes " & (ScrollSaved - ScrollBar1.Value)
    ScrollSaved = ScrollBar1.Value
End Sub

Private Sub ScrollBar1_Scroll()
    Label2.Caption = (ScrollSaved - ScrollBar1.Value) & " Widget Changes by
Scrolling"
End Sub
```

CanUndo, CanRedo Properties, UndoAction, RedoAction Methods Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpCanUndoCanRedoUndoActionRedoActionC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpCanUndoCanRedoUndoActionRedoActionA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpCanUndoCanRedoUndoActionRedoActionS"}

The following example demonstrates how to undo or redo text editing within a text box or within the text area of a **ComboBox**. This sample checks whether an undo or redo operation can occur and then performs the appropriate action. The sample uses the **CanUndo** and **CanRedo** properties, and the **UndoAction** and **RedoAction** methods.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **TextBox** named TextBox1.
- A **ComboBox** named ComboBox1.
- Two **CommandButton** controls named CommandButton1 and CommandButton2.

```
Private Sub CommandButton1_Click()
    If UserForm1.CanUndo = True Then
        UserForm1.UndoAction
        MsgBox "Undid IT"
    Else
        MsgBox "No undo performed."
    End If
End Sub

Private Sub CommandButton2_Click()
    If UserForm1.CanRedo = True Then
        UserForm1.RedoAction
        MsgBox "Redid IT"
    Else
        MsgBox "No redo performed."
    End If
End Sub

Private Sub UserForm_Initialize()
    TextBox1.Text = "Type your text here."

    ComboBox1.ColumnCount = 3
    ComboBox1.AddItem "Choice 1, column 1"
    ComboBox1.List(0, 1) = "Choice 1, column 2"
    ComboBox1.List(0, 2) = "Choice 1, column 3"

    CommandButton1.Caption = "Undo"
    CommandButton2.Caption = "Redo"
End Sub
```

BoundColumn Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpBoundColumnC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpBoundColumnA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpBoundColumnS"}

The following example demonstrates how the **BoundColumn** property influences the value of a **ListBox**. The user can choose to set the value of the **ListBox** to the index value of the specified row, or to a specified column of data in the **ListBox**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **ListBox** named ListBox1.
- A **Label** named Label1.
- Three **OptionButton** controls named OptionButton1, OptionButton2, and OptionButton3.

```
Private Sub UserForm_Initialize()
    ListBox1.ColumnCount = 2

    ListBox1.AddItem "Item 1, Column 1"
    ListBox1.List(0, 1) = "Item 1, Column 2"
    ListBox1.AddItem "Item 2, Column 1"
    ListBox1.List(1, 1) = "Item 2, Column 2"

    ListBox1.Value = "Item 1, Column 1"

    OptionButton1.Caption = "List Index"
    OptionButton2.Caption = "Column 1"
    OptionButton3.Caption = "Column 2"
    OptionButton2.Value = True
End Sub

Private Sub OptionButton1_Click()
    ListBox1.BoundColumn = 0
    Label1.Caption = ListBox1.Value
End Sub

Private Sub OptionButton2_Click()
    ListBox1.BoundColumn = 1
    Label1.Caption = ListBox1.Value
End Sub

Private Sub OptionButton3_Click()
    ListBox1.BoundColumn = 2
    Label1.Caption = ListBox1.Value
End Sub

Private Sub ListBox1_Click()
    Label1.Caption = ListBox1.Value
End Sub
```

List Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpListC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpListA"}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpListS"}

The following example swaps columns of a multicolumn **ListBox**. The sample uses the **List** property in two ways:

1. To access and exchange individual values in the **ListBox**. In this usage, **List** has subscripts to designate the row and column of a specified value.
1. To initially load the **ListBox** with values from an array. In this usage, **List** has no subscripts.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains a **ListBox** named ListBox1 and a **CommandButton** named CommandButton1.

```
Dim MyArray(6, 3)           'Array containing column values for ListBox.

Private Sub UserForm_Initialize()
    Dim i As Single

    ListBox1.ColumnCount = 3        'This list box contains 3 data columns

    'Load integer values MyArray
    For i = 0 To 5
        MyArray(i, 0) = i
        MyArray(i, 1) = Rnd
        MyArray(i, 2) = Rnd
    Next i

    'Load ListBox1
    ListBox1.List() = MyArray

End Sub


Private Sub CommandButton1_Click()
' Exchange contents of columns 1 and 3

    Dim i As Single
    Dim Temp As Single

    For i = 0 To 5
        Temp = ListBox1.List(i, 0)
        ListBox1.List(i, 0) = ListBox1.List(i, 2)
        ListBox1.List(i, 2) = Temp
    Next i
End Sub
```

ClientHeight, ClientLeft, ClientTop, ClientWidth Properties Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpTabStripC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpTabStripA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpTabStripS"}

The following example sets the dimensions of an **Image** to the size of a **TabStrip's** client area when the user clicks a **CommandButton**. This code sample uses the following properties: **Height**, **Left**, **Top**, **Width**, **ClientHeight**, **ClientLeft**, **ClientTop**, and **ClientWidth**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **CommandButton** named CommandButton1.
- A **TabStrip** named TabStrip1.
- An **Image** named Image1.

```
Private Sub UserForm_Initialize()
   CommandButton1.Caption = "Size Image to Tab Area"
   CommandButton1.WordWrap = True
   CommandButton1.AutoSize = True
End Sub


Private Sub CommandButton1_Click()
   Image1.ZOrder (fmFront)                  'Place Image in front of
TabStrip

   'ClientLeft and ClientTop are measured from the edge of the TabStrip,
   'not from the edges of the form containing the TabStrip.
   Image1.Left = TabStrip1.Left + TabStrip1.ClientLeft
   Image1.Top = TabStrip1.Top + TabStrip1.ClientTop
   Image1.Width = TabStrip1.ClientWidth
   Image1.Height = TabStrip1.ClientHeight

End Sub
```

MultiLine, WordWrap, ScrollBars Properties Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpMultiLineWordWrapScrollBarsC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpMultiLineWordWrapScrollBarsA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpMultiLineWordWrapScrollBarsS"}

The following example demonstrates the **MultiLine**, **WordWrap**, and **ScrollBars** properties on a **TextBox**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **TextBox** named TextBox1.
- Four **ToggleButton** controls named ToggleButton1 through ToggleButton4.

To see the entire text placed in the **TextBox**, set **MultiLine** and **WordWrap** to **True** by clicking the **ToggleButton** controls.

When **MultiLine** is **True**, you can enter new lines of text by pressing SHIFT+ENTER.

ScrollBars appears when you manually change the content of the **TextBox**.

```
Private Sub UserForm_Initialize()
'Initialize TextBox properties and toggle buttons

   TextBox1.Text = "Type your text here. Enter SHIFT+ENTER to move to a new
line."

   TextBox1.AutoSize = False
   ToggleButton1.Caption = "AutoSize Off"
   ToggleButton1.Value = False
   ToggleButton1.AutoSize = True

   TextBox1.WordWrap = False
   ToggleButton2.Caption = "WordWrap Off"
   ToggleButton2.Value = False
   ToggleButton2.AutoSize = True

   TextBox1.ScrollBars = 0
   ToggleButton3.Caption = "ScrollBars Off"
   ToggleButton3.Value = False
   ToggleButton3.AutoSize = True

   TextBox1.MultiLine = False
   ToggleButton4.Caption = "Single Line"
   ToggleButton4.Value = False
   ToggleButton4.AutoSize = True
 End Sub

Private Sub ToggleButton1_Click()
'Set AutoSize property and associated ToggleButton

   If ToggleButton1.Value = True Then
      TextBox1.AutoSize = True
      ToggleButton1.Caption = "AutoSize On"
   Else
      TextBox1.AutoSize = False
      ToggleButton1.Caption = "AutoSize Off"
   End If
```

```
End Sub

Private Sub ToggleButton2_Click()
'Set WordWrap property and associated ToggleButton

   If ToggleButton2.Value = True Then
      TextBox1.WordWrap = True
      ToggleButton2.Caption = "WordWrap On"
   Else
      TextBox1.WordWrap = False
      ToggleButton2.Caption = "WordWrap Off"
   End If
End Sub

Private Sub ToggleButton3_Click()
'Set ScrollBars property and associated ToggleButton

   If ToggleButton3.Value = True Then
      TextBox1.ScrollBars = 3
      ToggleButton3.Caption = "ScrollBars On"
   Else
      TextBox1.ScrollBars = 0
      ToggleButton3.Caption = "ScrollBars Off"
   End If
End Sub

Private Sub ToggleButton4_Click()
'Set MultiLine property and associated ToggleButton

   If ToggleButton4.Value = True Then
      TextBox1.MultiLine = True
      ToggleButton4.Caption = "Multiple Lines"
   Else
      TextBox1.MultiLine = False
      ToggleButton4.Caption = "Single Line"
   End If
End Sub
```

Drag-and-Drop Operation Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpDragandDropOperationC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpDragandDropOperationA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpDragandDropOperationS"}

The following example demonstrates a drag-and-drop operation from one **ListBox** to another using a **DataObject** to contain the dragged text. This code sample uses the **SetText** and **StartDrag** methods in the MouseMove event to implement the drag-and-drop operation.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains two **ListBox** controls named ListBox1 and ListBox2. You also need to add choices to the second **ListBox**.

```
Private Sub ListBox2_BeforeDragOver(ByVal Cancel As MSForms.ReturnBoolean,
ByVal Data As MSForms.DataObject, ByVal X As Single, ByVal Y As Single,
ByVal DragState As Long, ByVal Effect As MSForms.ReturnEffect, ByVal Shift
As Integer)
    Cancel = True
    Effect = 1
End Sub

Private Sub ListBox2_BeforeDropOrPaste(ByVal Cancel As
MSForms.ReturnBoolean, ByVal Action As Long, ByVal Data As
MSForms.DataObject, ByVal X As Single, ByVal Y As Single, ByVal Effect As
MSForms.ReturnEffect, ByVal Shift As Integer)
    Cancel = True
    Effect = 1
    ListBox2.AddItem Data.GetText
End Sub

Private Sub ListBox1_MouseMove(ByVal Button As Integer, ByVal Shift As
Integer, ByVal X As Single, ByVal Y As Single)
    Dim MyDataObject As DataObject
    If Button = 1 Then
        Set MyDataObject = New DataObject
        Dim Effect As Integer
        MyDataObject.SetText ListBox1.Value
        Effect = MyDataObject.StartDrag
    End If
End Sub

Private Sub UserForm_Initialize()
    For i = 1 To 10
        ListBox1.AddItem "Choice " & (ListBox1.ListCount + 1)
    Next i
End Sub
```

Accessing a Page Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpAccessingPageC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpAccessingPageA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpAccessingPageS"}

The following example accesses an individual page of a **MultiPage** in several ways:

- Using the **Pages** collection with a numeric index.
- Using the **Pages** collection with a string index.
- Using the **Pages** collection with the **Item** method.
- Using the name of the individual page in the **MultiPage**.
- Using the **SelectedItem** property.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains a **MultiPage** named MultiPage1.

```
Private Sub UserForm_Initialize()
    Dim PageNAme As String

    For i = 0 To MultiPage1.Count - 1
        'Use index (numeric or string)
        MsgBox "MultiPage1.Pages(i).Caption = " &
MultiPage1.Pages(i).Caption
        MsgBox "MultiPage1.Pages.Item(i).Caption = " &
MultiPage1.Pages.Item(i).Caption

        PageNAme = MultiPage1.Pages(i).Name
        MsgBox "PageName = " & PageNAme

        MsgBox "MultiPage1.Pages(PageName).Caption = " &
MultiPage1.Pages(PageNAme).Caption
        MsgBox "MultiPage1.Pages.Item(PageName).Caption = " &
MultiPage1.Pages.Item(PageNAme).Caption

        'Use Page object without referring to Pages collection
        If i = 0 Then
            MsgBox "MultiPage1.Page1.Caption= " & MultiPage1.Page1.Caption
        ElseIf i = 1 Then
            MsgBox "MultiPage1.Page2.Caption = " & MultiPage1.Page2.Caption
        End If

        'Use SelectedItem Property
        MultiPage1.Value = i
        MsgBox "MultiPage1.SelectedItem.Caption = " &
MultiPage1.SelectedItem.Caption
    Next i
End Sub
```

Adding a Control Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpAddingControlC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpAddingControlA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpAddingControlS"}

The following example uses the **Add** method to add a control to a form at run time and uses the AddControl event as confirmation that the control was added.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **CommandButton** named CommandButton1.
- A **Label** named Label1.

```
Dim Mycmd as Control
Private Sub CommandButton1_Click()

   Set Mycmd = Controls.Add("Forms.CommandButton.1") ', CommandButton2,
Visible)
   Mycmd.Left = 18
   Mycmd.Top = 150
   Mycmd.Width = 175
   Mycmd.Height = 20
   Mycmd.Caption = "This is fun." & Mycmd.Name

End Sub

Private Sub UserForm_AddControl(ByVal Control As MSForms.Control)
   Label1.Caption = "Control was Added."
End Sub
```

Adding a Control to a MultiPage Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpAddingControlToMultiPageC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpAddingControlToMultiPageA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpAddingControlToMultiPageS"}

The following example uses the **Add**, **Clear**, and **Remove** methods to add and remove a control to a **Page** of a **MultiPage** at run time.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **MultiPage** named MultiPage1.
- Three **CommandButton** controls named CommandButton1 through CommandButton3.

```
Dim MyTextBox As Control

Private Sub CommandButton1_Click()
Set MyTextBox = MultiPage1.Pages(0).Controls.Add("Forms.TextBox.1",
"MyTextBox", Visible)
End Sub

Private Sub CommandButton2_Click()
   MultiPage1.Pages(0).Controls.Clear
End Sub

Private Sub CommandButton3_Click()
   If MultiPage1.Pages(0).Controls.Count > 0  Then
      MultiPage1.Pages(0).Controls.Remove "MyTextBox"
   End If
End Sub

Private Sub UserForm_Initialize()
   CommandButton1.Caption = "Add control"
   CommandButton2.Caption = "Clear controls"
   CommandButton3.Caption = "Remove control"
End Sub
```

ListBox Control Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpListBoxC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpListBoxA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpListBoxS"}

The following example adds and deletes the contents of a **ListBox** using the **AddItem**, **RemoveItem**, and **SetFocus** methods, and the **ListIndex** and **ListCount** properties.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **ListBox** named ListBox1.
- Two **CommandButton** controls named CommandButton1 and CommandButton2.

```
Dim EntryCount As Single
Private Sub CommandButton1_Click()
    EntryCount = EntryCount + 1
    ListBox1.AddItem (EntryCount & " - Selection")
End Sub


Private Sub CommandButton2_Click()
    ListBox1.SetFocus

    'Ensure ListBox contains list items
    If ListBox1.ListCount >= 1 Then
        'If no selection, choose last list item.
        If ListBox1.ListIndex = -1 Then
            ListBox1.ListIndex = ListBox1.ListCount - 1
        End If
        ListBox1.RemoveItem (ListBox1.ListIndex)
    End If
End Sub


Private Sub UserForm_Initialize()
    EntryCount = 0
    CommandButton1.Caption = "Add Item"
    CommandButton2.Caption = "Remove Item"
End Sub
```

ZOrder Method Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpZOrderC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpZOrderA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpZOrderS"}

The following example sets the z-order of a **TextBox**, so the user can display the entire **TextBox** (by bringing it to the front of the z-order) or can place the **TextBox** behind other controls (by sending it to the back of the z-order).

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- Three **TextBox** controls named TextBox1 through TextBox3.
- A **ToggleButton** named ToggleButton1.

```
Private Sub ToggleButton1_Click()
If ToggleButton1.Value = True Then
   TextBox2.ZOrder (fmTop)          'Place TextBox2 on Top of z-order

   'Update ToggleButton caption to identify next state
   ToggleButton1.Caption = "Send TextBox2 to back"
Else
   TextBox2.ZOrder (1)              'Place TextBox2 on Bottom of z-order

   'Update ToggleButton caption to identify next state
   ToggleButton1.Caption = "Bring TextBox2 to front"
End If
End Sub


Private Sub UserForm_Initialize()
'Set up text boxes to show z-order in the form
TextBox1.Text = "TextBox 1"
TextBox2.Text = "TextBox 2"
TextBox3.Text = "TextBox 3"

TextBox1.Height = 40
TextBox2.Height = 40
TextBox3.Height = 40

TextBox1.Width = 60
TextBox2.Width = 60
TextBox3.Width = 60

TextBox1.Left = 10
TextBox1.Top = 10

TextBox2.Left = 25      'Overlap TextBox2 on TextBox1
TextBox2.Top = 25

TextBox3.Left = 40      'Overlap TextBox3 on TextBox2, TextBox1
TextBox3.Top = 40

ToggleButton1.Value = False
ToggleButton1.Caption = "Bring TextBox2 to Front"
ToggleButton1.Left = 10
ToggleButton1.Top = 90
ToggleButton1.Width = 50
```

```
ToggleButton1.Height = 50

End Sub
```

Accelerator, Caption Properties Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpAcceleratorCaptionC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpAcceleratorCaptionA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpAcceleratorCaptionS"}

This example changes the **Accelerator** and **Caption** properties of a **CommandButton** each time the user clicks the button by using the mouse or the accelerator key. The Click event contains the code to change the **Accelerator** and **Caption** properties.

To try this example, paste the code into the Declarations section of a form containing a **CommandButton** named CommandButton1.

```
Private Sub UserForm_Initialize()
   CommandButton1.Accelerator  = "C"         'Set Accelerator key to ALT +
C
End Sub

Private Sub CommandButton1_Click ()
   If CommandButton1.Caption = "OK" Then     'Check caption, then change
it.
      CommandButton1.Caption = "Clicked"
      CommandButton1.Accelerator  = "C"         'Set Accelerator key to ALT
+ C
   Else
      CommandButton1.Caption = "OK"
      CommandButton1.Accelerator  = "O"         'Set Accelerator key to ALT
+ O
   End If
End Sub
```

AutoSize Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpAutoSizeC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpAutoSizeA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpAutoSizeS"}

The following example demonstrates the effects of the **AutoSize** property with a single-line **TextBox** and a multiline **TextBox**. The user can enter text into either **TextBox** and turn **AutoSize** on or off independently of the contents of the **TextBox**. This code sample also uses the **Text** property.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- Two **TextBox** controls named TextBox1 and TextBox2.
- A **ToggleButton** named ToggleButton1.

```
Private Sub UserForm_Initialize()
    TextBox1.Text = "Single-line TextBox. Type your text here."

    TextBox2.MultiLine = True
    TextBox2.Text = "Multi-line TextBox. Type your text here. Use
CTRL+ENTER to start a new line."

    ToggleButton1.Value = True
    ToggleButton1.Caption = "AutoSize On"
    TextBox1.AutoSize = True
    TextBox2.AutoSize = True
End Sub

Private Sub ToggleButton1_Click()
    If ToggleButton1.Value = True Then
        ToggleButton1.Caption = "AutoSize On"
        TextBox1.AutoSize = True
        TextBox2.AutoSize = True
    Else
        ToggleButton1.Caption = "AutoSize Off"
        TextBox1.AutoSize = False
        TextBox2.AutoSize = False
    End If
End Sub
```

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

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpBoldC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpBoldA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpBoldS"}

The following example demonstrates a **Font** object and the **Bold**, **Italic**, **Size**, **StrikeThrough**, **Underline**, and **Weight** properties related to fonts. You can manipulate font properties of an object directly or by using an alias, as this example also shows.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **Label** named Label1.
- Four **ToggleButton** controls named ToggleButton1 through ToggleButton4.
- A second **Label** and a **TextBox** named Label2 and TextBox1.

```
Dim MyFont As StdFont

Private Sub ToggleButton1_Click()
    If ToggleButton1.Value = True Then
        MyFont.Bold = True            'Using MyFont alias to control font
        ToggleButton1.Caption = "Bold On"
        MyFont.Size = 22              'Increase the font size
    Else
        MyFont.Bold = False
        ToggleButton1.Caption = "Bold Off"
        MyFont.Size = 8               'Return font size to initial size
    End If

    TextBox1.Text = Str(MyFont.Weight)  'Bold and Weight are related
End Sub

Private Sub ToggleButton2_Click()
    If ToggleButton2.Value = True Then
        Label1.Font.Italic = True           'Using Label1.Font directly
        ToggleButton2.Caption = "Italic On"
    Else
        Label1.Font.Italic = False
        ToggleButton2.Caption = "Italic Off"
    End If
End Sub

Private Sub ToggleButton3_Click()
    If ToggleButton3.Value = True Then
        Label1.Font.Strikethrough = True            'Using Label1.Font
directly
        ToggleButton3.Caption = "StrikeThrough On"
    Else
        Label1.Font.Strikethrough = False
        ToggleButton3.Caption = "StrikeThrough Off"
    End If
End Sub

Private Sub ToggleButton4_Click()
    If ToggleButton4.Value = True Then
```

```
        MyFont.Underline = True            'Using MyFont alias for
Label1.Font
        ToggleButton4.Caption = "Underline On"
    Else
        Label1.Font.Underline = False
        ToggleButton4.Caption = "Underline Off"
    End If
End Sub

Private Sub UserForm_Initialize()
    Set MyFont = Label1.Font

    ToggleButton1.Value = True
    ToggleButton1.Caption = "Bold On"

    Label1.AutoSize = True   'Set size of Label1
    Label1.AutoSize = False

    ToggleButton2.Value = False
    ToggleButton2.Caption = "Italic Off"

    ToggleButton3.Value = False
    ToggleButton3.Caption = "StrikeThrough Off"

    ToggleButton4.Value = False
    ToggleButton4.Caption = "Underline Off"

    Label2.Caption = "Font Weight"
    TextBox1.Text = Str(Label1.Font.Weight)
    TextBox1.Enabled = False

End Sub
```

Border, Color Enhancements Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpBorderColorEnhancementsC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpBorderColorEnhancementsA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpBorderColorEnhancementsS"}

The following example demonstrates the **BorderStyle** and **SpecialEffect** properties, showing each border available through these properties. The example also demonstrates how to control color settings by using the **BackColor**, **BackStyle**, **BorderColor**, and **ForeColor** properties.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- Six **TextBox** controls named TextBox1 through TextBox6.
- Two **ToggleButton** controls named ToggleButton1 and ToggleButton2.

```
Private Sub UserForm_Initialize()
'Initialize each TextBox with a border style or special effect,
'and foreground and background colors

'TextBox1 initially uses a borderstyle
TextBox1.Text = "BorderStyle-Single"
TextBox1.BorderStyle = fmBorderStyleSingle
TextBox1.BorderColor = RGB(255, 128, 128)   'Color - Salmon
TextBox1.ForeColor = RGB(255, 255, 0)       'Color - Yellow
TextBox1.BackColor = RGB(0, 128, 64)        'Color - Green #2

'TextBoxes 2 through 6 initially use special effects
TextBox2.Text = "Flat"
TextBox2.SpecialEffect = fmSpecialEffectFlat
TextBox2.ForeColor = RGB(64, 0, 0)          'Color - Brown
TextBox2.BackColor = RGB(0, 0, 255)         'Color - Blue

'Ensure the background style for TextBox2 is initially opaque.
TextBox2.BackStyle = fmBackStyleOpaque

TextBox3.Text = "Etched"
TextBox3.SpecialEffect = fmSpecialEffectEtched
TextBox3.ForeColor = RGB(128, 0, 255)       'Color - Purple
TextBox3.BackColor = RGB(0, 255, 255)       'Color - Cyan

'Define BorderColor for later use (when borderstyle=fmBorderStyleSingle)
TextBox3.BorderColor = RGB(0, 0, 0)         'Color - Black

TextBox4.Text = "Bump"
TextBox4.SpecialEffect = fmSpecialEffectBump
TextBox4.ForeColor = RGB(255, 0, 255)       'Color - Magenta
TextBox4.BackColor = RGB(0, 0, 100)         'Color - Navy blue

TextBox5.Text = "Raised"
TextBox5.SpecialEffect = fmSpecialEffectRaised
TextBox5.ForeColor = RGB(255, 0, 0)         'Color - Red
TextBox5.BackColor = RGB(128, 128, 128)     'Color - Gray

TextBox6.Text = "Sunken"
TextBox6.SpecialEffect = fmSpecialEffectSunken
TextBox6.ForeColor = RGB(0, 64, 0)          'Color - Olive
TextBox6.BackColor = RGB(0, 255, 0)         'Color - Green #1
```

```
ToggleButton1.Caption = "Swap styles"
ToggleButton2.Caption = "Transparent/Opaque background"
End Sub

Private Sub ToggleButton1_Click()

'Swap borders between TextBox1 and TextBox3
If ToggleButton1.Value = True Then
   'Change TextBox1 from BorderStyle to Etched
   TextBox1.Text = "Etched"
   TextBox1.SpecialEffect = fmSpecialEffectEtched

   'Change TextBox3 from Etched to BorderStyle
   TextBox3.Text = "BorderStyle-Single"
   TextBox3.BorderStyle = fmBorderStyleSingle
Else
   'Change TextBox1 back to BorderStyle
   TextBox1.Text = "BorderStyle-Single"
   TextBox1.BorderStyle = fmBorderStyleSingle

   'Change TextBox3 back to Etched
   TextBox3.Text = "Etched"
   TextBox3.SpecialEffect = fmSpecialEffectEtched
End If
End Sub


Private Sub ToggleButton2_Click()

'Set background to Opaque or Transparent
If ToggleButton2.Value = True Then
   'Change TextBox2 to a transparent background
   TextBox2.BackStyle = fmBackStyleTransparent
Else
   'Change TextBox2 back to opaque background
   TextBox2.BackStyle = fmBackStyleOpaque
End If

End Sub
```

Column Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpColumnC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpColumnA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpColumnS"}

The following example loads a two-dimensional array with data and, in turn, loads two **ListBox** controls using the **Column** and **List** properties. Note that the **Column** property transposes the array elements during loading.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains two **ListBox** controls named ListBox1 and ListBox2.

```
Dim MyArray(6,3)

Private Sub UserForm_Initialize()
    Dim i As Single

    ListBox1.ColumnCount = 3        'The 1st list box contains 3 data
columns
    ListBox2.ColumnCount = 6        'The 2nd box contains 6 data columns

    'Load integer values into first column of MyArray
    For i = 0 To 5
        MyArray(i, 0) = i
    Next i

    'Load columns 2 and three of MyArray
    MyArray(0, 1) = "Zero"
    MyArray(1, 1) = "One"
    MyArray(2, 1) = "Two"
    MyArray(3, 1) = "Three"
    MyArray(4, 1) = "Four"
    MyArray(5, 1) = "Five"

    MyArray(0, 2) = "Zero"
    MyArray(1, 2) = "Un ou Une"
    MyArray(2, 2) = "Deux"
    MyArray(3, 2) = "Trois"
    MyArray(4, 2) = "Quatre"
    MyArray(5, 2) = "Cinq"

    'Load data into ListBox1 and ListBox2
    ListBox1.List() = MyArray
    ListBox2.Column() = MyArray

End Sub
```

ColumnWidths Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpColumnWidthsC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpColumnWidthsA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpColumnWidthsS"}

The following example uses the **ColumnWidths** property to change the column widths of a multicolumn **ListBox**. The example uses three **TextBox** controls to specify the individual column widths and uses the Exit event to specify the units of measure of each **TextBox**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **ListBox** named ListBox1.
- Three **TextBox** controls named TextBox1 through TextBox3.
- A **CommandButton** named CommandButton1.

Try entering the value 0 to hide a column.

```
Dim MyArray(2, 3) As String

Private Sub CommandButton1_Click()
   'ColumnWidths requires a value for each column separated by semicolons
   ListBox1.ColumnWidths = TextBox1.Text & ";" & TextBox2.Text & ";" &
TextBox3.Text
End Sub


Private Sub TextBox1_Exit(ByVal Cancel as MSForms.ReturnBoolean)
   'ColumnWidths accepts points (no units), inches or centimeters; make
inches the default
   If Not (InStr(TextBox1.Text, "in") > 0 Or InStr(TextBox1.Text, "cm") >
0) Then
      TextBox1.Text = TextBox1.Text & " in"
   End If
End Sub


Private Sub TextBox2_Exit(ByVal Cancel as MSForms.ReturnBoolean)
   'ColumnWidths accepts points (no units), inches or centimeters; make
inches the default
   If Not (InStr(TextBox2.Text, "in") > 0 Or InStr(TextBox2.Text, "cm") >
0) Then
      TextBox2.Text = TextBox2.Text & " in"
   End If
End Sub


Private Sub TextBox3_Exit(ByVal Cancel as MSForms.ReturnBoolean)
   'ColumnWidths accepts points (no units), inches or centimeters; make
   inches the default
   If Not (InStr(TextBox3.Text, "in") > 0 Or InStr(TextBox3.Text, "cm") >
0) Then
      TextBox3.Text = TextBox3.Text & " in"
   End If
End Sub


Private Sub UserForm_Initialize()
Dim i, j, Rows As Single
```

```
ListBox1.ColumnCount = 3
Rows = 2

For j = 0 To ListBox1.ColumnCount - 1
   For i = 0 To Rows - 1
      MyArray(i, j) = "Row " & i & ", Column " & j
   Next i
Next j

ListBox1.List() = MyArray           'Load MyArray into ListBox1

TextBox1.Text = "1 in"              '1-inch columns initially
TextBox2.Text = "1 in"
TextBox3.Text = "1 in"


End Sub
```

ControlTipText Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpControlTipTextC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpControlTipTextA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpControlTipTextS"}

The following example defines the **ControlTipText** property for three **CommandButton** controls and two **Page** objects in a **MultiPage**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **MultiPage** named MultiPage1.
- Three **CommandButton** controls named CommandButton1 through CommandButton3.

Note For an individual **Page** of a **MultiPage**, **ControlTipText** becomes enabled when the **MultiPage** or a control on the current page of the **MultiPage** has the focus.

```
Private Sub UserForm_Initialize()
   MultiPage1.Page1.ControlTipText = "Here in page 1"
   MultiPage1.Page2.ControlTipText = "Now in page 2"

   CommandButton1.ControlTipText = "And now here's"
   CommandButton2.ControlTipText = "a tip from"
   CommandButton3.ControlTipText = "your controls!"
End Sub
```

CurLine, CurTargetX, CurX, Text Properties Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpCurLineCurTargetXCurXC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpCurLineCurTargetXCurXA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpCurLineCurTargetXCurXS"}

The following example tracks the **CurLine**, **CurTargetX**, and **CurX** property settings in a multiline **TextBox**. These settings change in the KeyUp event as the user types into the **Text** property, moves the insertion point, and extends the selection using the keyboard.

To use this example, follow these steps:

1. Copy this sample code to the Declarations portion of a form.
1. Add one large **TextBox** named TextBox1 to the form.
1. Add three **TextBox** controls named TextBox2, TextBox3, and TextBox4 in a column.

```
Private Sub TextBox1_KeyUp(ByVal KeyCode As MSForms.ReturnInteger, ByVal
Shift As Integer)
   TextBox2.Text = TextBox1.CurLine
   TextBox3.Text = TextBox1.CurX
   TextBox4.Text = TextBox1.CurTargetX
End Sub


Private Sub UserForm_Initialize()
   TextBox1.MultiLine = True

   TextBox1.Text = "Type your text here. User CTRL + ENTER to start a new
line."
End Sub
```

HideSelection Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpHideSelectionC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpHideSelectionA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpHideSelectionS"}

The following example demonstrates the **HideSelection** property in the context of either a single form or more than one form. The user can select text in a **TextBox** and TAB to other controls on a form, as well as transfer the focus to a second form. This code sample also uses the **SetFocus** method, and the **EnterFieldBehavior**, **MultiLine**, and **Value** properties.

To use this example, follow these steps:

1. Copy this sample code (except for the last event subroutine) to the Declarations portion of a form.
1. Add a large **TextBox** named TextBox1, a **ToggleButton** named ToggleButton1, and a **CommandButton** named CommandButton1.
1. Insert a second form into this project named UserForm2.
1. Paste the last event subroutine of this listing into the Declarations section of UserForm2.
1. In this form, add a **CommandButton** named CommandButton1.
2. Run UserForm1.

```
' ***** Code for UserForm1 *****


Private Sub CommandButton1_Click()
   TextBox1.SetFocus
   UserForm2.Show                                'Bring up the second form.
End Sub


Private Sub ToggleButton1_Click()
   If ToggleButton1.Value = True Then
      TextBox1.HideSelection = False
      ToggleButton1.Caption = "Selection Visible"
   Else
      TextBox1.HideSelection = True
      ToggleButton1.Caption = "Selection Hidden"
   End If
End Sub


Private Sub UserForm_Initialize()
   TextBox1.MultiLine = True
   TextBox1.EnterFieldBehavior = fmEnterFieldBehaviorRecallSelection

'Fill the TextBox
   TextBox1.Text = "SelText indicates the starting point of selected text,
or the insertion point if no text is selected." _
      & Chr$(10) & Chr$(13) & "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. " _
      & Chr$(10) & Chr$(13) & "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."

   TextBox1.HideSelection = True
   ToggleButton1.Caption = "Selection Hidden"
   ToggleButton1.Value = False
```

```
End Sub
```

' ***** Code for UserForm2 *****

```
Private Sub CommandButton1_Click()
   UserForm2.Hide
End Sub
```

Picture, PicturePosition Properties Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpPicturePositionC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpPicturePositionA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpPicturePositionS"}

The following example uses a **ComboBox** to show the picture placement options for a control. Each time the user clicks a list choice, the picture and caption are updated on the **CommandButton**. This code sample also uses the **AddItem** method to populate the **ComboBox** choices.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **Label** named Label1.
- A **CommandButton** named CommandButton1.
- A **ComboBox** named ComboBox1.

```
Private Sub UserForm_Initialize()
   Label1.Left = 18
   Label1.Top = 12
   Label1.Height = 12
   Label1.Width = 190
   Label1.Caption = "Select picture placement relative to the caption."

   'Add list entries to combo box. The value of each entry matches the
   'corresponding ListIndex value in the combo box.
   ComboBox1.AddItem "Left Top"        'ListIndex = 0
   ComboBox1.AddItem "Left Center"     'ListIndex = 1
   ComboBox1.AddItem "Left Bottom"     'ListIndex = 2
   ComboBox1.AddItem "Right Top"       'ListIndex = 3
   ComboBox1.AddItem "Right Center"    'ListIndex = 4
   ComboBox1.AddItem "Right Bottom"    'ListIndex = 5

   ComboBox1.AddItem "Above Left"      'ListIndex = 6
   ComboBox1.AddItem "Above Center"    'ListIndex = 7
   ComboBox1.AddItem "Above Right"    'ListIndex = 8
   ComboBox1.AddItem "Below Left"      'ListIndex = 9
   ComboBox1.AddItem "Below Center"    'ListIndex = 10
   ComboBox1.AddItem "Below Right"    'ListIndex = 11

   ComboBox1.AddItem "Centered"        'ListIndex = 12

   ComboBox1.Style = fmStyleDropDownList  'Use drop-down list

   ComboBox1.BoundColumn = 0            'Combo box values are ListIndex
values
   ComboBox1.ListIndex = 0              'Set combo box to first entry


   ComboBox1.Left = 18
   ComboBox1.Top = 36
   ComboBox1.Width = 90
   ComboBox1.ListWidth = 90

   'Initialize CommandButton1
   CommandButton1.Left = 230
   CommandButton1.Top = 36
   CommandButton1.Height = 120
   CommandButton1.Width = 120
```

```
   'Note: Be sure to refer to a bitmap file that is present on your
   'Note: system, and to include the path in the filename.
   CommandButton1.Picture = LoadPicture("c:\windows\argyle.bmp")
   CommandButton1.PicturePosition = ComboBox1.Value
End Sub

Private Sub ComboBox1_Click()
   Select Case ComboBox1.Value
   Case 0  'Left Top
      CommandButton1.Caption = "Left Top"
      CommandButton1.PicturePosition = fmPicturePositionLeftTop

   Case 1  'Left Center
      CommandButton1.Caption = "Left Center"
      CommandButton1.PicturePosition = fmPicturePositionLeftCenter

   Case 2  'Left Bottom
      CommandButton1.Caption = "Left Bottom"
      CommandButton1.PicturePosition = fmPicturePositionLeftBottom

   Case 3  'Right Top
      CommandButton1.Caption = "Right Top"
      CommandButton1.PicturePosition = fmPicturePositionRightTop

   Case 4  'Right Center
      CommandButton1.Caption = "Right Center"
      CommandButton1.PicturePosition = fmPicturePositionRightCenter

   Case 5  'Right Bottom
      CommandButton1.Caption = "Right Bottom"
      CommandButton1.PicturePosition = fmPicturePositionRightBottom

   Case 6  'Above Left
      CommandButton1.Caption = "Above Left"
      CommandButton1.PicturePosition = fmPicturePositionAboveLeft

   Case 7  'Above Center
      CommandButton1.Caption = "Above Center"
      CommandButton1.PicturePosition = fmPicturePositionAboveCenter

   Case 8  'Above Right
      CommandButton1.Caption = "Above Right"
      CommandButton1.PicturePosition = fmPicturePositionAboveRight

   Case 9  'Below Left
      CommandButton1.Caption = "Below Left"
      CommandButton1.PicturePosition = fmPicturePositionBelowLeft

   Case 10 'Below Center
      CommandButton1.Caption = "Below Center"
      CommandButton1.PicturePosition = fmPicturePositionBelowCenter

   Case 11 'Below Right
      CommandButton1.Caption = "Below Right"
      CommandButton1.PicturePosition = fmPicturePositionBelowRight
```

```
   Case 12 'Centered
      CommandButton1.Caption = "Centered"
      CommandButton1.PicturePosition = fmPicturePositionCenter

   End Select

End Sub
```

Worksheet Binding Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpControlworksheetBindingC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpControlworksheetBindingA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpControlworksheetBindingS"}

The following example uses a range of worksheet cells in a **ListBox** and, when the user selects a row from the list, displays the row index in another worksheet cell. This code sample uses the **RowSource**, **BoundColumn**, and **ControlSource** properties.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains a **ListBox** named ListBox1. In the worksheet, enter data in cells A1:E4. You also need to make sure cell A6 contains no data.

```
Private Sub UserForm_Initialize()

ListBox1.ColumnCount = 5
ListBox1.RowSource = "a1:e4"

ListBox1.ControlSource = "a6"
ListBox1.BoundColumn = 0          'Place the ListIndex into cell a6
End Sub
```

Text Selection Properties Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpTextSelectionC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpTextSelectionA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpTextSelectionS"}

The following example tracks the selection-related properties (**SelLength**, **SelStart**, and **SelText**) that change as the user moves the insertion point and extends the selection using the keyboard. This example also uses the **Enabled** and **EnterFieldBehavior** properties.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- One large **TextBox** named TextBox1.
- Three **TextBox** controls in a column named TextBox2 through TextBox4.

```
Private Sub TextBox1_KeyUp(ByVal KeyCode As MSForms.ReturnInteger, ByVal
Shift As Integer)
   TextBox2.Text = TextBox1.SelStart
   TextBox3.Text = TextBox1.SelLength
   TextBox4.Text = TextBox1.SelText
End Sub


Private Sub UserForm_Initialize()
   TextBox1.MultiLine = True
   TextBox1.EnterFieldBehavior = fmEnterFieldBehaviorRecallSelection

   TextBox1.Text = "Type your text here. Use CTRL+ENTER to start a new
line."
End Sub
```

Accessing Controls Through the Controls Collection Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpControlsC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpControlsA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpControlsS"}

The following example accesses individual controls from the **Controls** collection using a For Each...Next loop. When the user presses CommandButton1, the other controls are placed in a column along the left edge of the form using the **Move** method.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains a **CommandButton** named CommandButton1 and several other controls.

```
Dim CtrlHeight As Single
Dim CtrlTop As Single
Dim CtrlGap As Single

Private Sub CommandButton1_Click()
   Dim MyControl As Control
   CtrlTop = 5

   For Each MyControl In Controls
      If MyControl.Name = "CommandButton1" Then
         'Don't move or resize this control.
      Else
         'Move method using named arguments
      MyControl.Move Top:=CtrlTop, Height:=CtrlHeight, Left:=5

      'Move method using unnamed arguments (left, top, width, height)
      'MyControl.Move 5, CtrlTop, ,CtrlHeight

         'Calculate top coordinate for next control
         CtrlTop = CtrlTop + CtrlHeight + CtrlGap
      End If
   Next

End Sub


Private Sub UserForm_Initialize()
   CtrlHeight = 20
   CtrlGap = 5

   CommandButton1.Caption = "Click to move controls"
   CommandButton1.AutoSize = True
   CommandButton1.Left = 120
   CommandButton1.Top = CtrlTop
End Sub
```

Cycle Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpCycleC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpCycleA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpCycleS"}

The following example defines the **Cycle** property for a **Frame** and two **Page** objects in a **MultiPage**.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- A **Frame** named Frame1.
- A **MultiPage** named MultiPage1 that contains two objects named Page1 and Page2.
- Two **CommandButton** controls named CommandButton1 and CommandButton2.

In the form, the **Frame**, and each **Page** of the **MultiPage**, place a couple of controls, so you can see how **Cycle** affects the tab order of the **Frame** and **MultiPage**.

The user should tab through the controls to observe how **Cycle** affects the tab order. Pressing CommandButton1 extends the tab order to include controls in the **Frame** and **Page** objects. Pressing CommandButton2 restricts the tab order.

```
Private Sub RestrictCycles()
'Limit tab order for the Frame and Page objects
    Frame1.Cycle = fmCycleCurrentForm
    MultiPage1.Page1.Cycle = fmCycleCurrentForm
    MultiPage1.Page2.Cycle = fmCycleCurrentForm
End Sub

Private Sub UserForm_Initialize()
   RestrictCycles
End Sub

Private Sub CommandButton1_Click()
'Extend tab order subforms (the Frame and Page objects)
    Frame1.Cycle = fmCycleAllForms
    MultiPage1.Page1.Cycle = fmCycleAllForms
    MultiPage1.Page2.Cycle = fmCycleAllForms
End Sub

Private Sub CommandButton2_Click()
   RestrictCycles
End Sub
```

Move Method Example for Controls Collection

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpControlsMoveC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpControlsMoveA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpControlsMoveS"}

The following example demonstrates moving all the controls on a form by using the **Move** method with the **Controls** collection. The user clicks on the **CommandButton** to move the controls.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains a **CommandButton** named CommandButton1 and several other controls.

```
Private Sub CommandButton1_Click()
    'Move each control on the form right 25 points and up 25 points.
Controls.Move 25, -25
End Sub
```

Parent Property Example

{ewc HLP95EN.DLL,DYNALINK,"See Also":"f3smpParentC"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"f3smpParentA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"f3smpParentS"}

The following example uses the **Parent** property to refer to the control or form that contains a specific control.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- Two **Label** controls named Label1 and Label2.
- A **CommandButton** named CommandButton1.
- One or more additional controls of your choice.

```
Dim MyControl As Object
Dim MyParent As Object
Dim ControlsIndex As Integer

Private Sub UserForm_Initialize()
    ControlsIndex = 0
    CommandButton1.Caption = "Get Control and Parent"
    CommandButton1.AutoSize = True
    CommandButton1.WordWrap = True
End Sub

Private Sub CommandButton1_Click()
    'Process Controls collection for UserForm
    Set MyControl = Controls.Item(ControlsIndex)
    Set MyParent = MyControl.Parent
    Label1.Caption = MyControl.Name
    Label2.Caption = MyParent.Name

    'Prepare index for next control on Userform
    ControlsIndex = ControlsIndex + 1
    If ControlsIndex >= Controls.Count Then
        ControlsIndex = 0
    End If
End Sub
```