**Topic Not Related**

There are no topics associated with this jump.

# Date and Time Example

This example uses a **CommandButton** Click event to update the **Caption** of a **Label** by choosing the value of two **CheckBoxes**.

The following controls and corresponding property values should be set:

- Add **CheckBox1**
  - Set **Caption** = "Show Date"
  - Set **Value** = 0
- Add **CheckBox2**
  - Set **Caption** = "Show Time"
  - Set **Value** = 0
- Add **Label1**
  - Set **Caption** = "Date and time displayed here"
  - Set **TextAlign** = Center
  - Set **BorderStyle** = Single
- Add **CommandButton1**
  - Set **Caption** = "Display"

For the **CommandButton1** Click event, add the following code:

```
Dim result
If CheckBox1.Value = True Then
    If CheckBox2.Value = True Then
        result = Date() & Chr(32) & Time()
    Else
        result = Date()
    End If
Else
    If CheckBox2.Value = True Then
        result = Time()
    Else
        result = "Date and time displayed here"
    End If
End If

Label1.Caption = result
```

# Form Information Example

This example uses **OptionButtons** to change the background color. of an HTML Layout and a **CommandButton** to display the width and height of the HTML Layout in a dialog box.

The following controls and corresponding property values should be set:

- Add **OptionButton1**
  - Set **Caption** = "Red"
  - Set **Value** = 0
- Add **OptionButton2**
  - Set **Caption** = "Green"
  - Set **Value** = 0
- Add **OptionButton3**
  - Set **Caption** = "Blue"
  - Set **Value** = 0
- Add **CommandButton1**
  - Set **Caption** = "Size"

For the following events, add the corresponding code:

- For the **OptionButton1** Click event:

```
Form.BackColor = RGB(255,0,0)
```

- For the **OptionButton2** Click event:

```
Form.BackColor = RGB(0,255,0)
```

- For the **Optionbutton3** Click event:

```
Form.BackColor = RGB(0,0,255)
```

- For the **CommandButton1** Click event:

```
MsgBox("HTML Layout width = " & Form.Width & chr(13) & chr(10) & _
"HTML Layoutheight = " & Form.Height)
```

# Adding Links via Colored Labels Example

This example uses **Labels** to provide links to other sites rather than underlined hypertext links. This text describes the process for adding one **Label**. This example could be extended to also use images and hot spots.

The following control and corresponding property values should be set:

- Add **Label1**
  - Select a background color and foreground color. Set Caption (for example "Microsoft").

For the **Label1** MouseDown event, add the following code:

```
Window.location.href = "http://www.microsoft.com"
```

# Hello World Example

This example uses a **CommandButton** Click event to display the message "Hello World" in a dialog box.

The following control and corresponding property value should be set:

- Add **CommandButton1**
  - Set **Caption** = "Push"

For the **CommandButton1** Click event, add the following code:

```
MsgBox("Hello, World!")
```

# Hide/Show Controls Example

This example demonstrates a method of hiding and showing **CommandButtons** on an HTML Layout.

The following controls and corresponding property values should be set:

- Add **CommandButton1**
  - Set Caption = "Show the other button"
- Add **CommandButton2**
  - Set Caption = "Bring back the first button"

For the following events, add the corresponding code:

- For the **CommandButton1** Click event:

```
CommandButton2.Visible = True
CommandButton1.Visible = False
```

- For the **CommandButton2** Click event:

```
CommandButton1.Visible = True
CommandButton2.Visible = False
```

# Add/Remove Items from a ListBox Example

This example demonstrates a method for interactively updating a **ListBox**.

The following controls and corresponding property values should be set:

- Add **CommandButton1**
  - Set **Caption** = "Add Item"
- Add **CommandButton2**
  - Set **Caption** = "Remove Item"
- Add **ListBox1**

For the following events, add the corresponding code:

- For the **CommandButton1** Click event:

```
Dim NewItem
NewItem = InputBox("Enter new item to add to list box","Add item")
NewItem = Trim(NewItem)
If Len(NewItem) > 0 Then
  ListBox1.AddItem(NewItem)
End If
```

- For the **CommandButton2** Click event

```
If ListBox1.ListIndex >= 0 Then
  ListBox1.RemoveItem(ListBox1.ListIndex)
  ListBox1.SetFocus
Else
  MsgBox("No item selected!")
End If
```

# Mouse Tracking Example

This example uses the MouseMove event for tracking mouse movement and updates a **Label** based on the mouse position.

The following controls and corresponding property values should be set:

- Add **Label1**
  - Set **Caption** = "Number One"
  - Set **BorderStyle** = "Single"
- Add **Label2**
  - Set **Caption** = "Number Two"
  - Set **BorderStyle** = "Single"
- Add **CommandButton1**
  - Set **Caption** = "Button 1"
- Add **Label3**
  - Set **Caption** = ""
  - Set **ID** = "lblDisplay"
  - Set **BorderStyle** = "Single"
  - Set **TextAlign** = "Center"

For the following events, add the corresponding code:

- For **Label1** MouseDown event:
```
lblDisplay.Caption = "Mouse down number one"
```
- For **Label1** MouseMove event:
```
lblDisplay.Caption = "Mouse moving over number one"
```
- For **Label2** MouseDown event:
```
lblDisplay.Caption = "Mouse down number two"
```
- For **Label2** MouseMove event:
```
lblDisplay.Caption = "Mouse moving over number two"
```
- For lblDisplay_MouseMove event:
```
lblDisplay.Caption = "Mouse moving over display label"
```
- For **CommandButton1** MouseMove event:
```
lblDisplay.Caption = "Mouse moving over command button"
```

# SpinButton Control Updating a Label Example

This example demonstrates dynamically updating the updating the **Caption** of a **Label** with a **SpinButton**.

The following controls and corresponding property values should be set:

- Add **SpinButton1**
- Add **Label1**
  - Set **Caption** = ""
  - Set **BorderStyle** = "Single"
  - Set **TextAlign** = "Center"
- For the **SpinButton1** event, add the following code:

```
Label1.Caption = SpinButton1.Value
```

# AfterUpdate Event



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

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

See Also Example Applies To

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

### Syntax

For TabStrip

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

For other controls

**Private Sub** *object*_**BeforeDragOver( ByVal** *Cancel* **As MSForms.ReturnBoolean**, **ByVal** *Data* **As DataObject**, **ByVal** *X* **As Single**, **ByVal** *Y* **As Single**, **ByVal** *DragState* **As fmDragState**, **ByVal** *Effect* **As MSForms.ReturnEffect**, **ByVal** *Shift* **As fmShiftState)**

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

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

See Also Example Applies To

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

### Syntax

For TabStrip

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

For other controls

**Private Sub** *object*_**BeforeDropOrPaste( ByVal** *Cancel* **As MSForms.ReturnBoolean**, **ByVal** *Action* **As fmAction**, **ByVal** *Data* **As DataObject**, **ByVal** *X* **As Single**, **ByVal** *Y* **As Single**, **ByVal** *Effect* **As MSForms.ReturnEffect**, **ByVal** *Shift* **As fmShiftState)**

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

| Part | Description |
|---|---|
| *object* | Required. A valid object name. |
| *index* | Required. The index of the control 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 should handle 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 **TabStrip**, VBScript initiates this event when it transfers a data object to the control.

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

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

# BeforeUpdate Event



Occurs before data in a control is changed.

### Syntax

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

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

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

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

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

# Change Event

See Also Example Applies To

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**.

### 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

See Also Example Applies To

Occurs in one of two cases:

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

### Syntax

For all controls
**Private Sub** *object*_**Click( )**

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

| Part | Description |
|---|---|
| *object* | Required. A valid object. |

### Remarks

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

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

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

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

1. MouseDown
1. MouseUp
2. Click

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

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

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

**Note** Left-clicking changes the value of a control, thus it initiates the Click event. Right-clicking does

not change the value of the control, so it does not initiate the Click event.

# DblClick Event

See Also Example Applies To

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

### Syntax

For TabStrip
**Private Sub** *object*_**DblClick(** *index* **As Long**, *Cancel* **As MSForms.ReturnBoolean)**
For other controls
**Private Sub** *object*_**DblClick(** *Cancel* **As MSForms.ReturnBoolean)**

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

| Part | Description |
|---|---|
| *object* | Required. A valid object. |
| *index* | Required. The position of a **Tab** object within a **Tabs** collection. |
| *Cancel* | Required. Event status. **False** indicates that the control should handle the event (default). **True** indicates the application should handle 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
1. MouseUp
2. Click
3. DblClick

If a control, such as **TextBox**, does not support Click, Click is omitted from 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 HTML Layout to ignore the second click, so clicking or double-clicking the button has the same effect.

# DropButtonClick Event



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



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

### Syntax

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

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

| Part | Description |
|---|---|
| *object* | Required. A valid object name. |
| *Cancel* | Required. Event status. **False** indicates that the control should handle the event (default). **True** indicates the application should handle the event and the focus should remain on the current control. |

### Remarks

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

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

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

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

# Error Event



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

### Syntax

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

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

| Part | Description |
|---|---|
| *object* | Required. A valid object name. |
| *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 that 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



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

### Syntax

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

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

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

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 an ActiveX Layout with a command button whose **Default** property is set to **True**.
- The user presses ESC on an ActiveX Layout 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
1. KeyPress
2. KeyUp

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

# KeyPress Event



Occurs when the user presses an ANSI key.

### Syntax

**Private Sub** *object*_**KeyPress(** *KeyANSI* **As MSForms.ReturnInteger)**

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

| Part | Description |
|---|---|
| *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 HTML Layout while the HTML Layout, or a control on it, has the focus. The event can occur either before or after the key is released.

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

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

A KeyPress event does not occur under the following conditions:

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

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

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

# MouseDown, MouseUp Events

See Also Example Applies To

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

### Syntax

For TabStrip

**Private Sub** *object*_**MouseDown(** *index* **As Long**, **ByVal** *Button* **As fmButton**, **ByVal** *Shift* **As fmShiftState**, **ByVal** *X* **As Single**, **ByVal** *Y* **As Single)**

**Private Sub** *object*_**MouseUp(** *index* **As Long**, **ByVal** *Button* **As fmButton**, **ByVal** *Shift* **As fmShiftState**, **ByVal** *X* **As Single**, **ByVal** *Y* **As Single)**

For other controls

**Private Sub** *object*_**MouseDown( ByVal** *Button* **As fmButton**, **ByVal** *Shift* **As fmShiftState**, **ByVal** *X* **As Single**, **ByVal** *Y* **As Single)**

**Private Sub** *object*_**MouseUp( ByVal** *Button* **As fmButton**, **ByVal** *Shift* **As fmShiftState**, **ByVal** *X* **As Single**, **ByVal** *Y* **As Single)**

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

| Part | Description |
|---|---|
| *object* | Required. A valid object. |
| *index* | Required. The index of the tab in a **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 HTML Layout. |

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

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

The sequence of mouse-related events is:

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

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

If a mouse button is pressed while the pointer is over an HTML Layout or control, that object will "capture" the mouse and receive 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 will receive 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* will be 6.

# MouseMove Event

See Also Example Applies To

Occurs when the user moves the mouse.

### Syntax

For TabStrip

**Private Sub** *object*_**MouseMove(** *index* **As Long**, **ByVal** *Button* **As fmButton**, **ByVal** *Shift* **As fmShiftState**, **ByVal** *X* **As Single**, **ByVal** *Y* **As Single)**

For other controls

**Private Sub** *object*_**MouseMove( ByVal** *Button* **As fmButton**, **ByVal** *Shift* **As fmShiftState**, **ByVal** *X* **As Single**, **ByVal** *Y* **As Single)**

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

| Part | Description |
|---|---|
| *object* | Required. A valid object name. |
| *index* | Required. The index of the tab in a **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 tab was clicked over. A –1 designates that the user did not click any of the tabs.

The settings for *Button* are:

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

The settings for *Shift* are:

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

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

| 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 HTML Layouts, controls on an HTML Layout, and labels.

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

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

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

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

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

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

# Scroll Event



Occurs when the scroll box is repositioned.

### Syntax

For ScrollBar
**Private Sub** *object*_**Scroll( )**

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

| Part | Description |
|---|---|
| *object* | Required. A valid object name. |

### 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 an ActiveX Layout return the following arguments: *ActionX*, *ActionY*, *ActualX*, and *ActualY*. *ActionX* and *ActionY* identify the action that occurred. *ActualX* and *ActualY* identify the distance 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 an ActiveX Layout. 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 when the user clicks on parts of the **ScrollBar** other than the scroll box.

# SpinDown, SpinUp Events



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 the **Value** property.

# CheckBox Control



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 will display a special mark (such as an X) and its current setting will be *Yes*, *True*, or *On*; if the user does not select the **CheckBox**, it will be empty and its setting will be *No*, *False*, or *Off*. Depending on the value of the **TripleState** property, a **CheckBox** can also have a Null value.

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

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

The default property of a **CheckBox** is the **Value** property. The initial value of the **Value** property is set to False.

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 may be able to use the **ListBox** instead of using a group of **CheckBox** controls.

# ComboBox Control



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

### Remarks

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

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

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

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

# CommandButton Control

See Also Example Properties Methods Events

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

**Remarks**

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

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

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

# Label Control

See Also Example Properties Methods Events

Displays descriptive text.

### Remarks

A **Label** control on an HTML Layout displays descriptive text such as titles, captions, pictures, or brief instructions. For example, labels for an address book might include a **Label** for a name, street, or city. A **Label** control 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.

# ListBox Control



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

**Remarks**

The **ListBox** control can either appear as a list or as a group of **OptionButton** controls, or **CheckBox** controls.

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

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

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

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

# OptionButton Control



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

**Remarks**

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

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

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

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

The default property for an **OptionButton** is the **Value** property. The initial value of the **Value** property is set to False.

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

# ScrollBar Control



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

**Remarks**

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

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

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

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

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

# SpinButton Control



Increases and decreases numbers.

**Remarks**

Clicking a **SpinButton** changes only the value of the **SpinButton**. You can write code that uses the **SpinButton** control 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** control to scroll through a range of values or a list of items, or to change the value displayed in a text box.

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

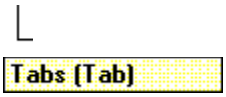
The default property for a **SpinButton** is the **Value** property.

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

# TabStrip Control

See Also Example Properties Methods Events

Presents a set of related controls as a visual group.




**Remarks**

You can use a **TabStrip** control 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



Displays information from a user.

### Remarks

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

Formatting applied to any piece of text in a **TextBox** control 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



Shows the selection state of an item.

**Remarks**

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

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

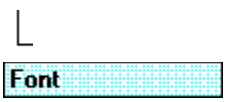
The default property of a **ToggleButton** is the **Value** property. The initial value of the **Value** property is set to False.

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

# Font Object



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




**Remarks**

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

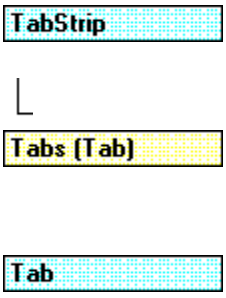
**Note** The font properties of your HTML Layout or container determine the default font attributes of controls you put on the HTML Layout.

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

# Tab Object



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

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

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

# Tabs Collection



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.

# Object Model for ActiveX Control Pad

See Also

The ActiveX Control Pad object model includes the following types of objects:

- Controls
- Objects (within collections)

Each element of the ActiveX Control Pad object model has some combination of properties, events, and methods that you can use to make your application work the way you want it to.

ActiveX Control Pad has two collections:

**Controls** collection—contains all the controls on a form.

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

# Accelerator Property

See Also Example Applies To

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 Properties window or in code. If the value of this property contains more than one character, the first character in the string becomes the value of **Accelerator**.

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

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

# Alignment Property

See Also Example Applies To

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

See Also Example Applies To

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. Specifies 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 cropped when they exceed the area of the control (default). |

### Remarks

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

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

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

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

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

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

# AutoTab Property

See Also Example Applies To

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

When the **AtuoTab** property is **True**, the **TabKeyBehavior** property is not in effect.

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

# AutoWordSelect Property

See Also Example Applies To

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 will include the entire word.

# BackColor Property

See Also Example Applies To

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 0 to 255. For example, you can specify teal blue as the integer value 4966415 or as red, green, and blue color components 15, 200, 75.

### Remarks

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

# BackStyle Property

See Also Example Applies To

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 an HTML Layout. If **BackStyle** is **fmBackStyleTransparent**, you can see through the control and look at anything on the HTML Layout located behind the control.

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

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

See Also Example Applies To

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

### Syntax

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

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

| Part | Description |
|---|---|
| *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.

There may be a difference between how a font appears on screen and how it looks printed, 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

See Also Example Applies To

Specifies the color of a control'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 a control. |

### Settings

You can use any integer that represents a valid color. You can also specify a color by using the RGB function with red, green, and blue color components. The value of each color component is an integer that ranges from 0 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

See Also Example Applies To

Specifies the type of border used by a control.

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

### Remarks

You can use either **BorderStyle** or **SpecialEffect** to specify the border for a control, but not both. If you specify a value other than zero 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 value other than zero for **SpecialEffect**, the system sets **BorderStyle** to zero.

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

# BoundColumn Property

See Also Example Applies To

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**.

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

See Also Example Applies To

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:<br>Null Indicates the item is in a null state, neither selected nor cleared.<br>–1 True. Indicates the item is selected.<br>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.<br>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 events 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.

# CanPaste Property

See Also Example Applies To

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

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

# Caption Property

See Also Example Applies To

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 an HTML Layout.

### Remarks

The text identifies or describes the object with which it is associated. For buttons and labels, the **Caption** property specifies the text that appears in the control. For **Tab** objects, it specifies the text that appears on the tab.

If a control's caption is too long, the caption is truncated. If an HTML Layout's caption is too long for the title bar, the title is displayed with an ellipsis.

The **ForeColor** property of the control determines the color of the text in the caption.

**Tip** If a control has both the **Caption** and **AutoSize** properties, setting **AutoSize** to **True** automatically adjusts the size of the control to frame the entire caption.

# ClientHeight, ClientLeft, ClientTop, ClientWidth Properties

See Also Example Applies To

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

See Also Example Applies To

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.

The **Column** property returns a *Variant* from the cursor. When a built-in cursor provides the value for *Variant* (such as when using the **AddItem** method), the value is a string. When an external cursor provides the value for *Variant*, formatting associated with the data is not included in the *Variant*.

### Remarks

You can use **Column** to assign the contents of a combo box or list box to another control, such as a text box.

If the user makes no selection when you refer to a column in a combo box or list box, the **Column** setting will be **Null**. You can check for this condition by using the **IsNull** function.

You can also use the **Column** property 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 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

See Also Example Applies To

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. 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

See Also Example Applies To

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

See Also Example Applies To

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 can 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 will be 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 will be left-aligned within the control and one or more of the rightmost columns will not be displayed. Users can scroll the list using the horizontal scroll bar to display the rightmost columns.

The minimum calculated column width is 72 points (1 inch). To produce columns narrower than this, you must specify the width explicitly.

Unless specified otherwise, column widths are measured in points. To specify another unit of measure, include the units as part of the values. The following examples specify column widths in several units of measure and describe how the various settings would fit in a three-column list box that is 4 inches wide.

| Setting | Effect |
|---|---|
| 90;72;90 | The first column is 90 points (1.25 inch); the second column is 72 points (1 inch); the third column is 90 points. |
| 6 cm;0;6 cm | The first column is 6 centimeters; the second column is hidden; the third column is 6 centimeters. Because part of the third column is visible, a horizontal scroll bar appears. |
| 1.5 in;0;2.5 in | The first column is 1.5 inches, the second column is hidden, and the third column is 2.5 inches. |
| 2 in;;2 in | The first column is 2 inches, the second column is 1 inch (default), and the third column is 2 inches. Because only half of the third column is visible, a horizontal scroll bar appears. |

| | |
|---|---|
| (Blank) | All three columns are the same width (1.33 inches). |

**Remarks**

In a combo box, the system displays the column designated by the **TextColumn** property in the text box portion of the control.

# Count Property

See Also Example Applies To

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** 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.

# CurLine Property

See Also Example Applies To

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

See Also Example Applies To

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 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

See Also Example Applies To

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.

# Delay Property

See Also Example Applies To

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** property. 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 the **Delay** property

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

See Also Example Applies To

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 will start a drag-and-drop operation on the selected text. If **DragBehavior** is disabled, dragging in a text box or combo box will select 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 the **DragBehavior** property to create a large number of text box styles.

# DropButtonStyle Property

See Also Example Applies To

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

See Also Example Applies To

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. Specifies 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 hot keys. The control is generally still accessible through code. |

### Remarks

Use the **Enabled** property to enable and disable controls. A disabled control appears dimmed, while an enabled control does not. Also, if a control displays a bitmap, the bitmap is dimmed whenever the control is dimmed. If **Enabled** is **False** for an **Image**, the control will not initiate events but it will also 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 it will appear normally (not dimmed) in the HTML Layout. The user can copy, but not edit, data in the control.
- If **Enabled** is **True** and **Locked** is **False**, the control can receive focus and it will appear normally in the HTML Layout. The user can copy and edit data in the control.
- If **Enabled** is **False** and **Locked** is **True**, the control cannot receive focus and it will appear dimmed in the HTML Layout. The user can neither copy nor edit data in the control.
- If **Enabled** and **Locked** are both **False**, the control cannot receive focus and it will appear dimmed in the HTML Layout. The user can neither copy nor edit data in the control.

You can combine the settings of the **Enabled** and the **TabStop** properties to prevent the user from selecting a command button with TAB, while still allowing the user to click the button. Setting **TabStop** to **False** means the command button will not appear in the tab order. However, if **Enabled** is **True**, then the user can still click the command button, as long as **TakeFocusOnClick** is set to **True**.

When the user tabs into an enabled **TabStrip**, the first page or tab in the control receives the focus. If the first page or tab of a **TabStrip** is disabled, the first enabled page or tab of that control will receive the focus. If all pages or tabs of a or **TabStrip** are disabled, the control will be disabled and will not be able to receive the focus.

# EnterFieldBehavior Property

See Also Example Applies To

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

See Also Example Applies To

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 will always move the focus to the next control in the tab order, regardless of the value of the **EnterKeyBehavior** property.

The effect of pressing CTRL+ENTER also depends on the value of **MultiLine**. If **MultiLine** is **True**, pressing CTRL+ENTER will create a new line regardless of the value of **EnterKeyBehavior**. If **MultiLine** is **False**, pressing CTRL+ENTER will have no effect.

# ForeColor Property

See Also Example Applies To

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 0 to 255. For example, you can specify teal blue as the integer value 4966415 or as red, green, and blue color components 15, 200, 75.

### Remarks

Use the **ForeColor** property for controls on HTML Layouts to make them easy to read or to convey a special meaning. For example, if a text box reports the number of units in stock, you can change the color of the text when the value falls below the reorder level.

For a **ScrollBar** or **SpinButton**, the **ForeColor** property sets the color of the arrows. For a **Font** object, the **ForeColor** property determines the color of the text.

# GroupName Property

See Also Example Applies To

Creates a group of mutually exclusive **OptionButton** controls.

### Syntax

*object*.**GroupName** [= *String*]

The **GroupName** 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

You can create buttons with transparent backgrounds, which can improve the visual appearance of your HTML Layout.

Clicking one button in a group sets all other buttons in the same group to **False**. All option buttons with the same **GroupName** within a single container are mutually exclusive. You can use the same group name in two containers, but doing so creates two groups (one in each container) rather than one group that includes both containers.

# Height, Width Properties

See Also Example Applies To

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 will store the new height or width. If you specify a setting for the **Left** or **Top** property that is less than zero, that value will be used to calculate the height or width of the control, but a portion of the control will not be visible on the HTML Layout.

If you move a control from one part of an HTML Layout to another, the setting of **Height** or **Width** will change only if you size the control as you move it. The settings of the control's **Left** and **Top** properties will change to reflect the control's new position relative to the edges of the HTML Layout that contains it.

The value assigned to **Height** or **Width** must be greater than or equal to zero. For most systems, the recommended range of values is from 0 to +32,767. Higher values may also work depending on your system configuration.

# HideSelection Property

See Also Example Applies To

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. Specifies 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 HTML Layout or a dialog box receives the focus, such as in a spell-checking procedure.

# ID Property

See Also Example Applies To

Specifies the name of a control or an object, or the name of a font to associate with a **Font** object.

### Syntax

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

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

| Part | Description |
|---|---|
| *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 the **ID** property, such as the maximum length of the name, vary from one application to another.

### Remarks

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

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

Each control added to an HTML Layout at design time must have a unique name.

For **Font** objects, the **ID** property identifies a particular typeface to use in the text portion of a control, object, or HTML Layout. The font's appearance on screen and in print may differ, depending on your computer and printer. If you select a font that your system can't display or that isn't installed, Windows will substitute a similar font.

# IMEMode Property

See Also Example Applies To

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 IME toolbar. The other is with a control's **IMEMode** property, which sets or returns the current mode of the IME. This property allows dynamic control of the IME through code.

The following example explains how **IMEMode** interacts with the IME toolbar. Assume that you have designed an HTML Layout with TextBox1 and CheckBox1. You have set TextBox1.IMEMode to 0, and you have set CheckBox1.IMEMode to 1. While in design mode you have used the IME toolbar to put

the IME in mode 2.

When you run the HTML Layout, the IME begins in mode 2. If you click TextBox1, the IME mode does not change because **IMEMode** for this control is 0. If you click CheckBox1, the IME will change to mode 1, because **IMEMode** for this control is 1. If you click again on TextBox1, the IME will remain in mode 1 (**IMEMode** is 0, so the IME retains its last setting).

However, you can override **IMEMode**. For example, assume you click CheckBox1 and the IME enters mode 1, as defined by **IMEMode** for the **CheckBox**. If you then use the IME toolbar to put the IME in mode 3, then the IME will be set to mode 3 when you click the control. This does not change the value of the property, it overrides the property until the next time you run the HTML Layout.

Every control makes a copy of the IME state in effect when that control receives focus. When it loses focus, it restores this saved state. This saving and restoring occurs without regard to the IMEMode property value of the control.

Controls, such as command buttons, that do not allow typing will disable the IME while they have focus.

Each change that a user makes to the IME while a control has focus immediately updates that control's **IMEMode** property, if it has one.

All controls will accept setting all IME modes without error, but when a mode is not "native" to a locale it will not be listed in property sheets in that locale, and it will have the same effect as a native mode. For example, the **fmIMEModeHangul** setting acts like the **fmIMEModeHiragana** setting if used in Japan.

All modes are native except:

In Japan, the Hangul modes are not native. Using them has the same effect as using Hiragana.

In Korea, the Hiragana and Katakana modes are not native. Using them has the same effect as Hangul or HangulFull, as appropriate.

In China, the Hiragana, Katakana, Hangul, and Alpha modes are not native. Using them has the same effect as On.

Everywhere else, the only native mode is **NoControl**. All other modes have the effect of **NoControl**.

# Index Property

See Also Example Applies To

The position of a **Tab** object within a **Tabs** 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 **Tabs** on a **TabStrip**. The index value for the first page or tab is 0; the index value of the second page or tab is 1, and so on.

In a **TabStrip**, the **Index** property refers to the tab only.

# IntegralHeight Property

See Also Example Applies To

Indicates whether a **ListBox** or **TextBox** displays full lines or partial lines of text in a list.

### Syntax

*object*.**IntegralHeight** [= *Boolean*]

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

| Part | Description |
|---|---|
| *object* | Required. A valid object. |
| *Boolean* | Optional. Specifies 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 will be automatically resized when necessary to show full rows. If **False**, the list will remain a fixed size; if items are taller than the available space in the list, the entire item will not be shown.

# LargeChange Property

See Also Example Applies To

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**.

# Left, Top Properties

See Also Example Applies To

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

### Syntax

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

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

| Part | Description |
|---|---|
| *object* | Required. A valid object. |
| *Single* | Optional. A numeric expression specifying the coordinates of an object. |

### Settings

Setting the **Left** or **Top** property to zero places the control's edge at the left or top edge of its container.

### Remarks

For most systems, the recommended range of values for **Left** and **Top** is from -32,767 to +32,767. Other values may also work depending on your system configuration. For a **ComboBox**, values of **Left** and **Top** apply to the text portion of the control, not to the list portion. When you move or size a control, its new **Left** setting is automatically entered in the Properties window. When you print an HTML Layout, the control's horizontal or vertical location is determined by its **Left** or **Top** setting.

# LineCount Property

See Also Example Applies To

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 have only one line.

# List Property

See Also Example Applies To

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 0 (zero). That is, the row number of the first row in the list is 0; the column number of the first column is also 0. The number of the second row or column is 1, and so on.

### Remarks

The **List** property works with the **ListCount** and **ListIndex** properties. Use **List** in code to access list items. A list is a variant array; each item in the list has a row number and a column number.

Initially, **ComboBox** and **ListBox** contain an empty list.

**Note** To specify items you want to display in a **ComboBox** or **ListBox**, use the **AddItem** method. To remove items, use the **RemoveItem** method.

You can also use **List** to copy an entire two-dimensional array of values to a control. This lets you quickly load a list of choices rather than using **AddItem** to individually load each element of the list.

# ListCount Property

See Also Example Applies To

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

See Also Example Applies To

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.

# ListRows Property

See Also Example Applies To

Specifies the maximum number of rows to display in the list before displaying a vertical scroll bar.

### Syntax

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

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

| 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 will appear at the right edge of the list box portion of the combo box.

# ListStyle Property

See Also Example Applies To

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 multiselect list (default). When the user selects an item from the group, the option button associated with that item is selected and the option buttons for the other items in the group are cleared. |

### Remarks

The **ListStyle** property lets you change the visual presentation of a **ListBox** or **ComboBox**. By specifying a setting other than **fmListStylePlain**, you can present the contents of either control as a group of individual items, with each item including a visual cue to indicate whether it is selected.

If the control supports a single selection (the **MultiSelect** property is set to **fmMultiSelectSingle**), the user can press one button in the group. If the control supports multiselect, the user can press two or more buttons in the group.

# ListWidth Property

See Also Example Applies To

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

See Also Example Applies To

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. Specifies 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

See Also Example Applies To

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 ActiveX Control Pad, this is true regardless of whether the list is sorted. This means the control finds the first occurrence that matches the entry, based on the order of items in the list. For example, entering either “abc” or “bc” will initiate the Click event for the following list:

```
abcde
bcdef
abcxyz
bchij
```

**Note** 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

See Also Example Applies To

Indicates whether the text that a user has typed into a **ComboBox** control 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 the **Value** property 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

See Also Example Applies To

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. Specifies 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

See Also Example Applies To

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

See Also Example Applies To

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

See Also Example Applies To

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

See Also Example Applies To

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 an HTML Layout. For example, the hourglass setting (11) is useful to indicate that the user must wait for a process or operation to finish.

Some icons vary depending on system settings, such as the icons associated with desktop themes.

# MultiLine Property

See Also Example Applies To

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. Specifies 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. 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

See Also Example Applies To

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. Specifies 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 determine 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 will occur 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

See Also Example Applies To

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.

# Orientation Property



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 the **SpinButton** or **ScrollBar** will appear horizontally or vertically. For example, if the control is wider than it is tall, the **SpinButton** or **ScrollBar** will appear horizontally; if the control is taller (WebBrowser Object)than it is wide, the **SpinButton** or **ScrollBar** will appear vertically.

# PasswordChar Property



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 will display the characters that the user types.

# Picture Property

See Also Example Applies To

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 an HTML Layout, you can use the control's property page to assign a bitmap to the **Picture** property. While running an HTML Layout, you must use the **LoadPicture** function to assign a bitmap to **Picture**.

To remove a picture that is assigned to a control, click the value of the **Picture** property in the property page and then press DELETE. Pressing BACKSPACE will not remove the picture.

**Note** For controls with captions, use the **PicturePosition** property to specify where to display the picture on the object.

Transparent pictures sometimes have a hazy appearance. If you do not like this appearance, display the picture on an **Image** control. **Image** controls support opaque images.

# PictureAlignment Property

See Also Example Applies To

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** will affect the tiling pattern. For example, if **PictureAlignment** is set to **fmPictureAlignmentUpperLeft**, the first copy of the image will be placed in the upper-left corner of the control or container and additional copies will be tiled from left to right across each row. If **PictureAlignment** is **fmPictureAlignmentCenter**, the first copy of the image will be placed at the center of the control or container, additional copies will be placed to the left and right to complete the row, and additional rows will be 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



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 will be relative to the center of the control.

This property is ignored if the **Picture** property does not specify a picture.

# PictureSizeMode Property

See Also Example Applies To

Specifies how to display the background picture on a control, HTML Layout, or HTML page.

### Syntax

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

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

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

### Settings

The settings for *fmPictureSizeMode* are:

| Constant | Value | Description |
|---|---|---|
| *fmPictureSizeModeClip* | 0 | Crops any part of the picture that is larger than the HTML Layout or HTML page (default). |
| *fmPictureSizeModeStretch* | 1 | Stretches the picture to fill the HTML Layout or HTML 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 HTML Layout or HTML page is smaller than the picture, this setting will show only the part of the picture that fits within the HTML Layout or HTML page.

The **fmPictureSizeModeStretch** and **fmPictureSizeModeZoom** settings both enlarge the image, but **fmPictureSizeModeStretch** causes distortion. The **fmPictureSizeModeStretch** setting enlarges the image horizontally and vertically until the image reaches the corresponding edges of the container or control. The **fmPictureSizeModeZoom** setting enlarges the image until it reaches either the horizontal or vertical edges of the container or control. If the image reaches the horizontal edges first, any remaining distance to the vertical edges will remain blank. If it reaches the vertical edges first, any remaining distance to the horizontal edges will remain blank.

# PictureTiling Property

See Also Example Applies To

Lets you tile a picture in an image control.

### Syntax

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

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

| Part | Description |
|---|---|
| *object* | Required. A valid object. |
| *Boolean* | Optional. Specifies 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

You can tile an image on an HTML Layout by drawing the **Image** the same size as the HTML Layout.

The tiling pattern depends on the current setting of the **PictureAlignment** and **PictureSizeMode** properties. For example, if **PictureAlignment** is set to **fmPictureAlignmentTopLeft**, the tiling pattern will start at the upper-left, repeating the picture across and down the height of the **Image**. If **PictureSizeMode** is set to **fmPictureSizeModeClip**, the tiling pattern will crop the last tile if it doesn't completely fit within the **Image**.

# ProportionalThumb Property



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. Specifies 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 will cover three-fourths of the scrolling region in the scroll bar.

If the scroll box is a fixed size, the system will determine its size based on the height and width of the scroll bar.

# ScrollBars Property

See Also Example Applies To

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 will always be visible, regardless of whether the object's contents fit within the its 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 **fmScrollBrsNone**, no scroll bar is displayed.
2. When **ScrollBars** is set to **fmScrollBarsHorizontal** or **fmScrollBarsBoth**, the control displays a horizontal scroll bar if the text is longer than the edit region and if the control has enough room to include the scroll bar underneath its edit region.
3. When **AutoSize** is **True**, the control enlarges itself to accommodate the addition of a scroll bar unless the control is at or near its maximum size.

For a multiline **TextBox**, you can display scroll bars by using the **ScrollBars**, **WordWrap**, and **AutoSize** properties. Scroll bars are hidden or displayed according to the following rules:

1. When **ScrollBars** is set to **fmScrollBarsNone**, no scroll bar is displayed.
2. When **ScrollBars** is set to **fmScrollBarsVertical** or **fmScrollBarsBoth**, the control displays a vertical scroll bar if the text is longer than the edit region and if the control has enough room to include the scroll bar at the right edge of its edit region.
3. When **WordWrap** is **True**, the multiline control will not display a horizontal scroll bar. Most multiline controls do not use a horizontal scroll bar.
4. A multiline control can display a horizontal scroll bar if the following conditions occur

simultaneously:

- The edit region contains a word that is longer than the edit region's width.
- The control has enabled horizontal scroll bars.
- The control has enough room to include the scroll bar under the edit region.
- The **WordWrap** property is set to **False**.

# Selected Property

See Also Example Applies To

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. Specifies 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 multiselect list box. You can also use this property to select or deselect rows in a list from code.

The default value of this property is based on the current selection state of the **ListBox**.

For single-selection list boxes, the **Value** or **ListIndex** properties are recommended for getting and setting the selection. In this case, **ListIndex** returns the index of the selected item. However, in a multiple selection, **ListIndex** returns the index of the row contained within the focus rectangle, regardless of whether the row is actually selected.

When a list box control's **MultiSelect** property is set to **None**, only one row can have its **Selected** property set to **True**.

Entering a value that is out of range for the index does not generate an error message, but it also does not set a property for any item in the list.

# SelectedItem Property

See Also Example Applies To

Returns or sets the currently selected **Tab** object.

### Syntax

*object*.**SelectedItem**

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

| Part | Description |
|---|---|
| *object* | Required. A valid **TabStrip**. |

### Remarks

Use the **SelectedItem** property to programmatically control the currently selected **Tab** object. For example, you can use **SelectedItem** to assign values to properties of a **Tab** object.

# SelectionMargin Property



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. Specifies 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 will also be printed.

# SelLength Property

See Also Example Applies To

Specifies the number of characters selected in a text box or the text portion of a combo box.

### Syntax

*object*.**SelLength** [= *Long*]

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

| Part | Description |
|---|---|
| *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 the **SelLength** property to zero.

The default value, zero, means that no text is currently selected.

# SelStart Property



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 0. |

### 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

See Also Example Applies To

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

See Also Example Applies To

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

See Also Example Applies To

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

See Also Example Applies To

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 value other than zero 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 **TabStrip**, identifies the style of the tabs on the control.

### Syntax

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

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

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

See Also Example Applies To

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**. |

### Settings

If the value is zero, tab widths will be 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 zero, all tabs will have an identical width as specified by this property.

### Remarks

The minimum size is 4 points.

# TabIndex Property

See Also Example Applies To

Specifies the position of a single object in the HTML Layout's tab order.

### Syntax

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

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

| Part | Description |
|---|---|
| *object* | Required. A valid object. |
| *Integer* | Optional. An integer from 0 to one less than the number of controls on the HTML Layout 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



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 will always move 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 the **MultiLine** property. If **MultiLine** is **True**, pressing CTRL+TAB will create a new line regardless of the value of **TabKeyBehavior**. If **MultiLine** is **False**, pressing CTRL+TAB will have no effect.

# TabOrientation Property



Specifies the location of the tabs on a **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

See Also Example Applies To

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. Specifies 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. |

### Remarks

The **TabStop** property can be set only at design time.

# Text Property

See Also Example Applies To

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

### Syntax

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

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

| Part | Description |
|---|---|
| *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) will be set to the row that matches **Text**. If the value of **Text** does not match a row, **ListIndex** will be set to –1.

For a **ListBox**, the value of the **Text** property must match an existing list entry. Specifying a value that does not match an existing list entry causes an error.

You cannot use the **Text** property 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

See Also Example Applies To

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 affects only the edit region. The **TextAlign** 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

See Also Example Applies To

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



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



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.

# TripleState Property



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. Specifies 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 supports True and False only (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

See Also Example Applies To

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:<br>Null Indicates the item is in a null state, neither selected nor cleared.<br>–1 True. Indicates the item is selected.<br>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**. |
| **TextBox** | The text in the edit region. |

### Remarks

For a **CommandButton**, setting the **Value** property to **True** in a procedure initiates the button's Click event.

For a **ComboBox**, changing the contents of **Value** does not change the value of **BoundColumn**. To add or delete entries in a **ComboBox**, you can use the **AddItem** or **RemoveItem** method.

The **Value** property cannot be used with a multiselect list box.

# Visible Property

See Also Example Applies To

Specifies whether a control 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

To hide an object at startup, set the **Visible** property to **False** at design time. Setting this property in code enables you to hide and later redisplay a control at run time in response to a particular event.

All controls are visible at design time.

# WordWrap Property

See Also Example Applies To

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. Specifies 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**.

# Add Method

See Also Example Applies To

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

### Syntax

For TabStrip

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

For other controls

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

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

| Part | Description |
|---|---|
| *object* | Required. A valid object name. |
| *Name* | Optional. Specifies the name of the object being added. If a name is not specified, the system will generate a default name based on the rules of the application where the HTML Layout is used. |
| *Caption* | Optional. Specifies the caption to appear on a tab or a control. If a caption is not specified, the system will generate a default caption based on the rules of the application where the HTML Layout is used. |
| *index* | Optional. Identifies the position of a tab within a **Tabs** collection. If an index is not specified, the system will append the tab to the end of the **Tabs** collection and will assign the appropriate index value. |
| *ProgID* | Required. Programmatic identifier. A text string with no spaces that identifies an object class. The standard syntax of 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: 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 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**.

| Control | ProgID value |
|---|---|
| **CheckBox** | Forms.CheckBox.1 |
| **ComboBox** | Forms.ComboBox.1 |
| **CommandButton** | Forms.CommandButton.1 |
| **Image** | Forms.Image.1 |
| **Label** | Forms.Label.1 |
| **ListBox** | Forms.ListBox.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 **TabStrip**, the **Add** method returns a **Tab** object. The index value for the first **Tab** of a collection is 0. The value for the second **Tab** is 1, and so on.

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

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

```
userform1.thebox.text
```

If you add a control 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 **ID** property at run time only if you added that control at run time with the **Add** method.

# AddItem Method

See Also Example Applies To

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 will place the item or row at that position within the list. If you omit *varIndex*, the method will add 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 will fail.

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

# Clear Method

See Also Example Applies To

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

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

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

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

# Copy Method

See Also Example Applies To

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 **TextBox** or **ComboBox**, the **Copy** method copies the currently selected text.

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

# Cut Method

See Also Example Applies To

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

# DropDown Method



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



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* | Optional. An integer or string specifying the data format to use when pasting the Clipboard contents. |

### Settings

The settings for *format* are:

| Constant | Value | Description |
|---|---|---|
| *fmCFText* | 1 | Text format. |

### Remarks

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

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

# GetFromClipboard, GetText Methods



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

### Syntax

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

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

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

### Settings

The settings for *format* are:

| Constant | Value | Description |
|---|---|---|
| *fmCFText* | 1 | Text format. |

### Remarks

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

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

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

# Item Method

See Also Example Applies To

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

See Also Example Applies To

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

### Syntax

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

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

| 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. |

### Settings

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

### Remarks

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

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

# Paste Method

See Also Example Applies To

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 an HTML Layout, you can paste any object onto the HTML Layout.

# PutInClipboard Method



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 **DataObject** and the Clipboard both support multiple formats, but they support only one data item of a given format.

For example, the **DataObject** might include one text item stored using the Clipboard format. If the Clipboard contains data in the same format as new data, the new data will replace the existing data on the Clipboard. If the new data is in a new format, the new data and the new format will both be added to the Clipboard.

# Remove Method

See Also Example Applies To

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

### Syntax

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

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

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

See Also Example Applies To

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. |

# SetFocus Method



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 will revert to the previous object and an error will be generated.

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

# SetText Method



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

### Syntax

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

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

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

### Settings

The settings for *format* are:

| Constant | Value | Description |
|---|---|---|
| *fmCFText* | 1 | Text format. |

### Remarks

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

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

# StartDrag Method



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 button. The effect of the drag-and-drop operation depends on the effect chosen for the drop target.

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

# ZOrder Method

See Also Example Applies To

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

This method does not affect the 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.

**ANSI character set**

American National Standards Institute (ANSI) 8-bit character set used by Microsoft Windows to represent up to 256 characters (0–255) using your keyboard. The first 128 characters (0–127) correspond to the letters and symbols on a standard U.S. keyboard. The second 128 characters (128–255) represent special characters, such as letters in international alphabets, accents, currency symbols, and fractions.

**array**

A set of sequentially indexed elements having the same intrinsic data type. Each element of an array has a unique identifying index number. Changes made to one element of an array do not affect the other elements.

**class**

The formal definition of an object. The class acts as the template from which an instance of an object is created at run time. The class defines the properties of the object and the methods used to control the object's behavior.

**container**

An object that can contain other objects.

**module**

A set of declarations followed by procedures.

**named arguments**

An argument that has a name that is predefined in the object library. Instead of providing a value for each argument in a specified order expected by the syntax, you can use named arguments to assign values in any order. For example, suppose a method accepts three arguments:

**DoSomeThing *namedarg1***, ***namedarg2***, ***namedarg3***

By assigning values to named arguments, you can use the following statement:

```
DoSomeThing namedarg3 := 4, namedarg2 := 5, namedarg1 := 20
```

Note that the arguments don't need to appear in their normal positional order.

**Null**

A value indicating that a variable contains no valid data. **Null** is the result of an explicit assignment of **Null** to a variable or any operation between expressions that contains **Null**.

**point**

In typography, a point is 1/72 inch. The size of a font is usually expressed in points.

**project**

A set of modules.

**tab order**

The order in which the focus moves from one field or object to the next as you press TAB or SHIFT+TAB.

**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 will be processed when the event is initiated. Also called keyboard accelerator, shortcut key, keyboard shortcut.

**background color**

The color of the client region of an empty window or display screen, on which all drawing and color display takes place.

**class identifier (CLSID)**

A unique identifier (UUID) that identifies an object. An object registers its CLSID in the system registration database so the object can be loaded and programmed by other applications.

**clear**

To change a setting to "off" or remove a value.

**client region**

The portion of a window where an application displays output such as text or graphics.

**collection**

An object that contains a set of related objects. An object's position in the collection can change whenever a change occurs in the collection; therefore, the position of any specific object in the collection may vary.

**context ID**

A unique number or string that corresponds to a specific object in an application. Context IDs are used to create links between the application and corresponding Help topics.

**control group**

A set of controls that are conceptually or logically related. Controls that are conceptually related are usually viewed together but do not necessarily affect each other. Controls that are logically related affect each other. For example, setting one button in a group of option buttons will set the value of all other buttons in the group to False.

**control tip**

A brief phrase that describes a control, a **Page**, or a **Tab**. The control tip appears when the user briefly holds the mouse pointer over a control without clicking. A control tip is similar to a ToolTip. ActiveX Control Pad provides ToolTips to developers at design time, while developers provide control tips to end-users at run time.

**cursor**

A piece of software that returns rows of data to the application. A cursor on a result set indicates the current position in the result set.

**cycle**

To move through a group of objects in a defined order.

**data format**

The structure or appearance of a unit of data, such as a file, a database record, a cell in a spreadsheet, or text in a word-processing document.

**dominant control**

A reference for the **Align** command and **Make Same Size** command on the **Format** menu. When aligning controls, the selected controls align to the dominant control. When sizing controls, the selected controls are assigned the dimensions of the dominant control.

The dominant control is indicated by white sizing handles. The sizing handles of the other selected controls are black.

**drop source**

The selected text or object that is dragged in a drag-and-drop operation.

**focus**

The ability to receive mouse clicks or keyboard input at any one time. In Microsoft Windows, only one window, HTML Layout, or control can have this ability at a time. The object that "has the focus" is usually indicated by a highlighted caption or title bar. The focus can be set by the user or by the application.

**foreground color**

The color that is currently selected for drawing or displaying text on screen. In monochrome displays, the foreground color is the color of a bitmap or other graphic.

**grid block**

The space between two adjacent grid points.

**Input Method Editor (IME)**

An application that translates what you type into characters of a DBCS language, such as Japanese or Chinese. As the user types, the IME displays possible equivalents. The user selects the most appropriate entry.

**inherited property**

A property that has acquired the characteristics of another class.

**keyboard state**

A return value that identifies which keys are pressed and whether the keyboard modifiers SHIFT, CTRL, and ALT are pressed.

**OLE container control**

A Visual Basic control that is used to link and embed objects from other applications in a Visual Basic application.

**OLE object**

An object in an application that can be linked or embedded.

**OLE status code**

The error number portion of a data structure that returns information for error conditions. The data structure is defined by Object Linking and Embedding.

**placeholder**

A character that masks or hides another character for security reasons. For example, when a user types a password, an asterisk is displayed on the screen to take the place of each character typed.

**property page**

A grouping of properties presented as a tabbed page of a Properties Window.

**RGB**

A color value system used to describe colors as a mixture of red (R), green (G), and blue (B). The color is defined as a set of three integers (R,G,B) where each integer ranges from 0–255. A value of 0 indicates a total absence of a color component. A value of 255 indicates the highest intensity of a color component.

**SendKeys statement**

Sends one or more keystrokes to the active window as if typed at the keyboard.

**single-precision value**

Single (single-precision floating-point) variables are stored as IEEE 32-bit (4-byte) floating-point numbers, ranging in value from -3.402823E38 to -1.401298E-45 for negative values and from 1.401298E-45 to 3.402823E38 for positive values. The type-declaration character for Single is !.

**system colors**

Colors defined by the operating system for a specific type of monitor and video adapter. Each color is associated with a specific part of the user interface, such as a window title or a menu.

**target**

An object onto which the user drops the object being dragged.

**transparent**

Describes the background of the object if the background is not visible. Instead of the background, you see whatever is behind the object, such as an image or picture used as a backdrop in your application. Use the **BackStyle** property to make the background transparent.

**z-order**

The visual layering of controls on an HTML Layout along the z-axis (depth). The z-order determines which controls are in front of other controls.