Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

LeaveCell Event

SeeAlso Example Applies To

Occurs immediately before the currently active cell changes to a different cell.

Syntax

Private Sub *object*_**LeaveCell()**

The LeaveCell event syntax has one part:

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

Remarks

This event is used to validate the contents of a cell.

This event does not occur when moving focus to a different control.

Visual Studio 6.0

Visual Basic: MSChart Control

LegendActivated Event

See Also Example Applies To

Occurs when the user double clicks on the chart legend.

Syntax

Private Sub *object*_**LegendActivated** (*mouseFlags* **As Integer**, *cancel* **As Integer**)

The LegendActivated event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
mouseFlags	Integer. Indicates whether a key is held down when the mouse button is clicked, as described in Settings.
cancel	Integer. This argument is not used at this time.

Settings

The event handler determines if a key is held down when the mouse button is clicked and sets *mouseFlags* to:

Constants	Description
VtChMouseFlagShiftKeyDown	If the SHIFT key is held down.
VtChMouseFlagControlKeyDown	If the CONTROL key is held down.

Visual Studio 6.0

Visual Basic: MSChart Control

LegendSelected Event

See Also Example Applies To

Occurs when the user clicks on the chart legend.

Syntax

Private Sub *object*_**LegendSelected** (*mouseFlags* **As Integer**, *cancel* **As Integer**)

The LegendSelected event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
mouseFlags	Integer. Indicates whether a key is held down when the mouse button is clicked.
cancel	Integer. This argument is not used at this time.

Settings

The event handler determines if a key is held down when the mouse button is clicked and sets *mouseFlags* to:

Constants	Description
VtChMouseFlagShiftKeyDown	If the SHIFT key is held down.
VtChMouseFlagControlKeyDown	If the CONTROL key is held down.

Visual Studio 6.0

Visual Basic: MSChart Control

LegendUpdated Event

See Also Example Applies To

Occurs when the chart legend has changed.

Syntax

Private Sub *object*_**LegendUpdated** (*updateFlags* **As Integer**)

The LegendUpdated event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
updateFlags	Integer. Provides information about the update of the legend, as described in Settings.

Settings

The following table lists the constants for *updateFlags*.

Constant	Description
VtChNoDisplay	Absence of update flags; the chart display is not affected. (Defined as 0.)
VtChDisplayPlot	Update will cause the plot to repaint.
VtChLayoutPlot	Update will cause the plot to lay out.
VtChDisplayLegend	Update will cause the legend to repaint.
VtChLayoutLegend	Update will cause the legend to lay out.
VtChLayoutSeries	Update will cause the series to lay out.
VtChPositionSection	A chart section has been moved or resized.

Visual Basic Reference

Visual Studio 6.0

LinkClose Event

See Also Example Applies To

Occurs when a DDE conversation terminates. Either application in a DDE conversation may terminate a conversation at any time.

Syntax

```
Private Sub Form_LinkClose( )

Private Sub MDIForm_LinkClose( )

Private Sub object_LinkClose([index As Integer])
```

The LinkClose event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	An integer that uniquely identifies a control if it's in a control array.

Remarks

Typically, you use a LinkClose event procedure to notify the user that a DDE conversation has been terminated. You can also include troubleshooting information on reestablishing a connection or where to go for assistance. For brief messages, use the **MsgBox** function.

Visual Basic Reference

Visual Studio 6.0

LinkError Event

See Also Example Applies To

Occurs when there is an error during a DDE conversation. This event is recognized only as the result of a DDE-related error that occurs when no Visual Basic code is being executed. The error number is passed as an argument.

Syntax

```
Private Sub Form_LinkError(linkerr As Integer)

Private Sub MDIForm_LinkError(linkerr As Integer)

Private Sub object_LinkError([index As Integer,]linkerr As Integer)
```

The LinkError event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
linkerr	Error number of the DDE-related error, as described in Return Values.
index	An integer that uniquely identifies a control if it's in a control array.

Return Values

The following table lists all error numbers returned for the *linkerr* argument and a brief explanation of each error:

Value	Description
1	The other application has requested data in the wrong format. This error may occur several times in succession as Visual Basic tries to find a format the other application recognizes.
6	The destination application attempted to continue a DDE conversation after you set the **LinkMode** property on your source form to 0 (None).
7	All the source links are in use (there is a limit of 128 links per source).
8	For destination controls: An automatic link or **LinkRequest** method failed to update the data in the control. For source forms: The destination attempted to poke data to a control and the attempt failed.

11	Not enough memory for DDE.

Remarks

Use a LinkError event procedure to notify the user of the particular error that has occurred. You can also include code to fix the problem or troubleshooting information on reestablishing a connection or on where to go for assistance. For brief messages, use the **MsgBox** function.

Visual Basic Reference

LinkError Event Example

This example is attached to a **TextBox** control, MyTextBox, that handles selected errors. The procedure displays a message (adapted from the error list in the LinkError event topic) based on the error number passed as the argument `LinkErr`. You can adapt this code to a source form by substituting `Form_LinkError` for `MyTextBox_LinkError.` This example is for illustration only.

```
Private Sub MyTextBox_LinkError (LinkErr As Integer)
Dim Msg
Select Case LinkErr
   Case 1
      Msg = "Data in wrong format."
   Case 11
      Msg = "Out of memory for DDE."
End Select
MsgBox Msg, vbExclamation, "MyTextBox"
End Sub
```

Visual Basic Reference

Visual Studio 6.0

LinkExecute Event

See Also Example Applies To

Occurs when a command string is sent by a destination application in a DDE conversation. The destination application expects the source application to perform the operation described by the string.

Syntax

```
Private Sub object_LinkExecute(cmdstr As String, cancel As Integer)
```

The LinkExecute event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
cmdstr	The command string expression sent by the destination application.
cancel	An integer that tells the destination whether the command string was accepted or refused. Setting *cancel* to 0 informs the destination that the command string was accepted. Setting *cancel* to any nonzero value informs the destination that the command string was rejected. (The default is set to -1, indicating *cancel*.)

Remarks

There is no required syntax for *cmdstr*. How your application responds to different strings is completely up to you.

If you haven't created a LinkExecute event procedure, Visual Basic rejects command strings from destination applications.

Visual Basic Reference

LinkExecute Event Example

This example defines a set of commands for destinations to use in DDE conversations to which your application will respond. This example is for illustration only.

```
Private Sub Form_LinkExecute (CmdStr As String, Cancel As Integer)
   Cancel = False
   Select Case LCase(CmdStr)
   Case "{big}"
      WindowState = 2   ' Maximize window.
   Case "{little}"
      WindowState = 1   ' Minimize window.
   Case "{hide}"
      Visible = False   ' Hide form.
   Case "{view}"
      Visible = True      ' Display form.
   Case Else
      Cancel = True      ' Execute not allowed.
   End Select
End Sub
```

Visual Basic Reference

Visual Studio 6.0

LinkNotify Event

See Also Example Applies To

Occurs when the source has changed the data defined by the DDE link if the **LinkMode** property of the destination control is set to 3 (Notify).

Syntax

```
Private Sub object_LinkNotify([index As Integer])
```

The LinkNotify event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	An integer that uniquely identifies a control if it's in a control array.

Remarks

Typically, in the LinkNotify event your code notifies the user, gets the new data immediately, or defers getting the data until later. You can use the **LinkRequest** method to obtain the new data from the source.

Visual Basic Reference

LinkNotify Event Example

This example is attached to a **PictureBox** control, Picture1, that has its **LinkTopic** and **LinkItem** properties set to specify a graphic in the source, and its **LinkMode** property set to 3 (Notify). When the source changes this data, the procedure updates the **PictureBox** control immediately only if the **PictureBox** is on the active form; otherwise, it sets a flag variable. This example is for illustration only.

```
Private Sub Picture1_LinkNotify ()
   If Screen.ActiveForm Is Me Then
   Picture1.LinkRequest      ' Picture is on active form, so update.
   Else
      NewDataFlag = True   ' Assumed to be a module-level variable.
   End If
End Sub
```

Visual Basic Reference

Visual Studio 6.0

LinkOpen Event

See Also Example Applies To

Occurs when a DDE conversation is being initiated.

Syntax

```
Private Sub Form_LinkOpen(cancel As Integer)

Private Sub MDIForm_LinkOpen(cancel As Integer)

Private Sub object_LinkOpen([index As Integer,]cancel As Integer)
```

The LinkOpen event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
cancel	An integer that determines whether the DDE conversation is established or not. Leaving *cancel* set to 0 (the default) establishes the conversation. Setting *cancel* to any nonzero value refuses the conversation.
index	An integer that uniquely identifies a control if it's in a control array.

Remarks

This event occurs for forms when a destination application is initiating a DDE conversation with the form. It occurs for controls when a control is initiating a DDE conversation with a source application.

Visual Basic: Page Designer

Visual Studio 6.0

Load Event (DHTMLPage)

See Also Example Applies To

Occurs when the browser loads an HTML page in your application.

Syntax

Private Sub *object*_**Load**

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

Remarks

There are two distinct behaviors for this event, depending on whether your page is set to load asynchronously or not.

- When loading asynchronously, the Load event fires after the first element on the page is created.
- When loading synchronously, the Load event fires after all elements have been created.

Programmers can use this event when running synchronously (when the **AsyncLoad** property is set to **False**) as a notification that all elements have been loaded onto the page.

Visual Basic Reference

Visual Studio 6.0

Load Event

See Also Example Applies To

Occurs when a form is loaded. For a startup form, occurs when an application starts as the result of a **Load** statement or as the result of a reference to an unloaded form's properties or controls.

Syntax

```
Private Sub Form_Load( )

Private Sub MDIForm_Load( )
```

Remarks

Typically, you use a Load event procedure to include initialization code for a form for example, code that specifies default settings for controls, indicates contents to be loaded into **ComboBox** or **ListBox** controls, and initializes form-level variables.

The Load event occurs after the Initialize event.

When you reference a property of an unloaded form in code, the form is automatically loaded but isn't automatically made visible unless the **MDIChild** property is set to **True**. If an **MDIForm** object isn't loaded and an MDI child form is loaded, both the **MDIForm** and the child form are automatically loaded and both become visible. Other forms aren't shown until you either use the **Show** method or set the **Visible** property to **True**.

The following code in an **MDIForm** Load event automatically loads an MDI child form (assuming `Form1` has its **MDIChild** property set to **True**):

```
Dim NewForm As New Form1
NewForm.Caption = "New Form"    ' Loads form by reference.
```

Because all child forms become visible when loaded, the reference to the **Caption** property loads the form and makes it visible.

Note When you create procedures for related events, such as Activate, GotFocus, Paint, and Resize, be sure that their actions don't conflict and that they don't cause recursive events.

Visual Basic Reference

Load Event Example

This example loads items into a **ComboBox** control when a form is loaded. To try this example, paste the code into the Declarations section of a form that contains a **ComboBox**, and then press F5.

```
Private Sub Form_Load ()
   Combo1.AddItem "Mozart"      ' Add items to list.
   Combo1.AddItem "Beethoven"
   Combo1.AddItem "Rock 'n Roll"
   Combo1.AddItem "Reggae"
   Combo1.ListIndex = 2         ' Set default selection.
End Sub
```

Visual Basic Reference

Visual Studio 6.0

LostFocus Event (UserControl Object and UserDocument Object)

See Also Example Applies To

Occurs in the object or constituent control when focus leaves it.

Syntax

Sub *object*_**LostFocus()**

The LostFocus event syntax has these parts:

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

Remarks

This LostFocus event is not the same LostFocus extender event that the developer who uses *object* handles. This LostFocus event is for the author of *object*, and is internal to *object*.

This event is useful if *object* needs to know that the focus is now on it.

Object itself can get focus only when the **CanGetFocus** property is **True** and there are no constituent controls that can receive focus.

The LostFocus event is raised before the ExitFocus event.

Visual Basic Reference

Visual Studio 6.0

LostFocus Event

See Also Example Applies To

Occurs when an object loses the focus, either by user action, such as tabbing to or clicking another object, or by changing the focus in code using the **SetFocus** method.

Syntax

```
Private Sub Form_LostFocus( )

Private Sub object_LostFocus([index As Integer])
```

The LostFocus event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	An integer that uniquely identifies a control if it's in a control array.

Remarks

A LostFocus event procedure is primarily useful for verification and validation updates. Using LostFocus can cause validation to take place as the user moves the focus from the control. Another use for this type of event procedure is enabling, disabling, hiding, and displaying other objects as in a GotFocus event procedure. You can also reverse or change conditions that you set up in the object's GotFocus event procedure.

If an .exe file built by Visual Basic displays a dialog box created by a .dll file also built in Visual Basic, the .exe file's form will get Deactivate and LostFocus events. This may be unexpected, because you should not get the Deactivate event:

- If the object is an out-of-process component.
- If the object isn't written in Visual Basic.
- In the development environment when calling a DLL built in Visual Basic.

Visual Basic Reference

LostFocus Event Example

This example changes the color of a **TextBox** control when it receives or loses the focus (selected with the mouse or TAB key) and displays the appropriate text in the **Label** control. To try this example, paste the code into the Declarations section of a form that contains two **TextBox** controls and a **Label** control, and then press F5 and move the focus between Text1 and Text2.

```
Private Sub Text1_GotFocus ()
   ' Show focus with red.
   Text1.BackColor = RGB(255, 0, 0)
   Label1.Caption = "Text1 has the focus."
End Sub

Private Sub Text1_LostFocus ()
   ' Show loss of focus with blue.
   Text1.BackColor = RGB(0, 0, 255)
   Label1.Caption = "Text1 doesn't have the focus."
End Sub
```

Visual Basic Reference

Visual Studio 6.0

MouseDown, MouseUp Events (ActiveX Controls)

See Also Example Applies To

Occur when the user presses (MouseDown) or releases (MouseUp) a mouse button.

Syntax

Private Sub Form_MouseDown(*button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single)**

Private Sub MDIForm_MouseDown(*button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single)**

Private Sub *object*_**MouseDown(**[*index* **As Integer**,]*button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single)**

Private Sub Form_MouseUp(*button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single)**

Private Sub MDIForm_MouseUp(*button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single)**

Private Sub *object* _**MouseUp(**[*index* **As Integer**,]*button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single)**

The MouseDown and MouseUp event syntaxes have these parts:

Part	Description
object	Returns an object expression that evaluates to an object in the Applies To list.
index	Returns an integer that uniquely identifies a control if it's in a control array.
button	Returns an integer that identifies the button that was pressed (MouseDown) or released (MouseUp) to cause the event. The *button* argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Only one of the bits is set, indicating the button that caused the event.
shift	Returns an integer that corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the *button* argument is pressed or released. A bit is set if the key is down. The *shift* argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The *shift* argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT were pressed, the value of *shift* would be 6.
x, *y*	Returns a number that specifies the current location of the mouse pointer. The *x* and *y* values are always expressed in terms of the coordinate system set by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties of the object.

Remarks

Use a MouseDown or MouseUp event procedure to specify actions that will occur when a given mouse button is pressed or released. Unlike the Click and DblClick events, 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.

The following applies to both Click and DblClick events:

- If a mouse button is pressed while the pointer is over a form or control, that object "captures" the mouse and receives all mouse events up to and including the last MouseUp event. This implies that the *x*, *y* mouse-pointer coordinates returned by a mouse event may not always be in the internal area of the object that receives them.
- If mouse buttons are pressed in succession, the object that captures the mouse after the first press receives all mouse events until all buttons are released.

If you need to test for the *button* or *shift* arguments, you can use constants listed in the Visual Basic (VB) object library in the Object Browser to define the bits within the argument:

Constant (Button)	Value	Description
vbLeftButton	1	Left button is pressed
vbRightButton	2	Right button is pressed
vbMiddleButton	4	Middle button is pressed

Constant (Shift)	Value	Description
vbShiftMask	1	SHIFT key is pressed.
vbCtrlMask	2	CTRL key is pressed.
vbAltMask	4	ALT key is pressed.

The constants then act as bit masks you can use to test for any combination of buttons without having to figure out the unique bit field value for each combination.

Note You can use a MouseMove event procedure to respond to an event caused by moving the mouse. The *button* argument for MouseDown and MouseUp differs from the *button* argument used for MouseMove. For MouseDown and MouseUp, the *button* argument indicates exactly one button per event, whereas for MouseMove, it indicates the current state of all buttons.

Visual Basic Reference

Visual Studio 6.0

MouseDown, MouseUp Events

See Also Example Applies To

Occur when the user presses (MouseDown) or releases (MouseUp) a mouse button.

Syntax

Private Sub Form_MouseDown(*button* **As Integer,** *shift* **As Integer,** *x* **As Single,** *y* **As Single)**

Private Sub MDIForm_MouseDown(*button* **As Integer,** *shift* **As Integer,** *x* **As Single,** *y* **As Single)**

Private Sub *object*_**MouseDown(**[*index* **As Integer,**]*button* **As Integer,** *shift* **As Integer,** *x* **As Single,** *y* **As Single)**

Private Sub Form_MouseUp(*button* **As Integer,** *shift* **As Integer,** *x* **As Single,** *y* **As Single)**

Private Sub MDIForm_MouseUp(*button* **As Integer,** *shift* **As Integer,** *x* **As Single,** *y* **As Single)**

Private Sub *object* **_MouseUp(**[*index* **As Integer,**]*button* **As Integer,** *shift* **As Integer,** *x* **As Single,** *y* **As Single)**

The MouseDown and MouseUp event syntaxes have these parts:

Part	Description
object	Returns an object expression that evaluates to an object in the Applies To list.
index	Returns an integer that uniquely identifies a control if it's in a control array.
button	Returns an integer that identifies the button that was pressed (MouseDown) or released (MouseUp) to cause the event. The *button* argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Only one of the bits is set, indicating the button that caused the event.
shift	Returns an integer that corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the *button* argument is pressed or released. A bit is set if the key is down. The *shift* argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The *shift* argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT were pressed, the value of *shift* would be 6.
x, *y*	Returns a number that specifies the current location of the mouse pointer. The *x* and *y* values are always expressed in terms of the coordinate system set by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties of the object.

Remarks

Use a MouseDown or MouseUp event procedure to specify actions that will occur when a given mouse button is pressed or released. Unlike the Click and DblClick events, 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.

The following applies to both Click and DblClick events:

- If a mouse button is pressed while the pointer is over a form or control, that object "captures" the mouse and receives all mouse events up to and including the last MouseUp event. This implies that the *x*, *y* mouse-pointer coordinates returned by a mouse event may not always be in the internal area of the object that receives them.
- If mouse buttons are pressed in succession, the object that captures the mouse after the first press receives all mouse events until all buttons are released.

If you need to test for the *button* or *shift* arguments, you can use constants listed in the Visual Basic (VB) object library in the Object Browser to define the bits within the argument:

Constant (Button)	Value	Description
vbLeftButton	1	Left button is pressed
vbRightButton	2	Right button is pressed
vbMiddleButton	4	Middle button is pressed

Constant (Shift)	Value	Description
vbShiftMask	1	SHIFT key is pressed.
vbCtrlMask	2	CTRL key is pressed.
vbAltMask	4	ALT key is pressed.

The constants then act as bit masks you can use to test for any combination of buttons without having to figure out the unique bit field value for each combination.

Note You can use a MouseMove event procedure to respond to an event caused by moving the mouse. The *button* argument for MouseDown and MouseUp differs from the *button* argument used for MouseMove. For MouseDown and MouseUp, the *button* argument indicates exactly one button per event, whereas for MouseMove, it indicates the current state of all buttons.

Visual Basic Reference

MouseDown, MouseUp Events Example

This example demonstrates a simple paint application. The MouseDown event procedure works with a related MouseMove event procedure to enable painting when any mouse button is pressed and dragged. The MouseUp event procedure disables painting. To try this example, paste the code into the Declarations section of a form; and then press F5, click the form, and move the mouse while the mouse button is pressed.

```
Dim PaintNow As Boolean
Private Sub Form_MouseDown (Button As Integer, Shift As Integer, X As
Single, Y As Single)
   PaintNow = True   ' Enable painting.
End Sub

Private Sub Form_MouseUp (Button As Integer, Shift As Integer, X As Single, Y As Single)
   PaintNow = False   ' Disable painting.
End Sub

Private Sub Form_MouseMove (Button As Integer, Shift As Integer, X As Single, Y As Single)
   If PaintNow Then
      PSet (X, Y)      ' Draw a point.
   End If
End Sub

Private Sub Form_Load ()
   DrawWidth = 10      ' Use wider brush.
   ForeColor = RGB(0, 0, 255)   ' Set drawing color.
End Sub
```

Visual Basic Reference

Visual Studio 6.0

MouseMove Event (ActiveX Controls)

See Also Example Applies To

Occurs when the user moves the mouse.

Syntax

Private Sub Form_MouseMove(*button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single)**

Private Sub MDIForm_MouseMove(*button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single)**

Private Sub *object*_**MouseMove(**[*index* **As Integer**,] *button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single)**

The MouseMove event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	An integer that uniquely identifies a control if it's in a control array.
button	An integer that corresponds to the state of the mouse buttons in which a bit is set if the button is down. The *button* argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. It indicates the complete state of the mouse buttons; some, all, or none of these three bits can be set, indicating that some, all, or none of the buttons are pressed.
shift	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys. A bit is set if the key is down. The *shift* argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The *shift* argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT were pressed, the value of *shift* would be 6.
x, *y*	A number that specifies the current location of the mouse pointer. The *x* and *y* values are always expressed in terms of the coordinate system set by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties of the object.

Remarks

The MouseMove event is 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.

If you need to test for the *button* or *shift* arguments, you can use constants listed in the Visual Basic (VB) object library in the Object Browser to define the bits within the argument:

Constant (Button)	Value	Description
vbLeftButton	1	Left button is pressed.
vbRightButton	2	Right button is pressed.
vbMiddleButton	4	Middle button is pressed.

Constant (Shift)	Value	Description
vbShiftMask	1	SHIFT key is pressed.
vbCtrlMask	2	CTRL key is pressed.
vbAltMask	4	ALT key is pressed.

The constants then act as bit masks you can use to test for any combination of buttons without having to figure out the unique bit field value for each combination.

You test for a condition by first assigning each result to a temporary integer variable and then comparing the *button* or *shift* arguments to a bit mask. Use the **And** operator with each argument to test if the condition is greater than zero, indicating the key or button is pressed, as in this example:

```
LeftDown = (Button And vbLeftButton) > 0
CtrlDown = (Shift And vbCtrlMask) > 0
```

Then, in a procedure, you can test for any combination of conditions, as in this example:

```
If LeftDown And CtrlDown Then
```

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

The *button* argument for MouseMove differs from the *button* argument for MouseDown and MouseUp. For MouseMove, the *button* argument indicates the current state of all buttons; a single MouseMove event can indicate that some, all, or no buttons are pressed. For MouseDown and MouseUp, the *button* argument indicates exactly one button per event.

Any time you move a window inside a MouseMove event, it can cause a cascading event. MouseMove events are generated when the window moves underneath the pointer. A MouseMove event can be generated even if the mouse is perfectly stationary.

Visual Basic Reference

Visual Studio 6.0

MouseMove Event

See Also Example Applies To

Occurs when the user moves the mouse.

Syntax

```
Private Sub Form_MouseMove(button As Integer, shift As Integer, x As Single, y As Single)

Private Sub MDIForm_MouseMove(button As Integer, shift As Integer, x As Single, y As Single)

Private Sub object_MouseMove([index As Integer,] button As Integer, shift As Integer, x As Single, y As Single)
```

The MouseMove event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	An integer that uniquely identifies a control if it's in a control array.
button	An integer that corresponds to the state of the mouse buttons in which a bit is set if the button is down. The *button* argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. It indicates the complete state of the mouse buttons; some, all, or none of these three bits can be set, indicating that some, all, or none of the buttons are pressed.
shift	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys. A bit is set if the key is down. The *shift* argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The *shift* argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT were pressed, the value of *shift* would be 6.
x, *y*	A number that specifies the current location of the mouse pointer. The *x* and *y* values are always expressed in terms of the coordinate system set by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties of the object.

Remarks

The MouseMove event is 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.

If you need to test for the *button* or *shift* arguments, you can use constants listed in the Visual Basic (VB) object library in the Object Browser to define the bits within the argument:

Constant (Button)	Value	Description
vbLeftButton	1	Left button is pressed.
vbRightButton	2	Right button is pressed.
vbMiddleButton	4	Middle button is pressed.

Constant (Shift)	Value	Description
vbShiftMask	1	SHIFT key is pressed.
vbCtrlMask	2	CTRL key is pressed.
vbAltMask	4	ALT key is pressed.

The constants then act as bit masks you can use to test for any combination of buttons without having to figure out the unique bit field value for each combination.

You test for a condition by first assigning each result to a temporary integer variable and then comparing the *button* or *shift* arguments to a bit mask. Use the **And** operator with each argument to test if the condition is greater than zero, indicating the key or button is pressed, as in this example:

```
LeftDown = (Button And vbLeftButton) > 0
CtrlDown = (Shift And vbCtrlMask) > 0
```

Then, in a procedure, you can test for any combination of conditions, as in this example:

```
If LeftDown And CtrlDown Then
```

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

The *button* argument for MouseMove differs from the *button* argument for MouseDown and MouseUp. For MouseMove, the *button* argument indicates the current state of all buttons; a single MouseMove event can indicate that some, all, or no buttons are pressed. For MouseDown and MouseUp, the *button* argument indicates exactly one button per event.

Any time you move a window inside a MouseMove event, it can cause a cascading event. MouseMove events are generated when the window moves underneath the pointer. A MouseMove event can be generated even if the mouse is perfectly stationary.

Visual Basic Reference

MouseMove Event Example

This example demonstrates a simple paint application. The MouseDown event procedure works with a related MouseMove event procedure to enable painting when any mouse button is pressed. The MouseUp event procedure disables painting. To try this example, paste the code into the Declarations section of a form, and then press F5 and click the form and move the mouse while the mouse button is pressed.

```
Dim PaintNow As Boolean   ' Declare variable.
Private Sub Form_MouseDown (Button As Integer, Shift As Integer, X As Single, Y As Single)
   PaintNow = True   ' Brush on.
End Sub

Private Sub Form_MouseUp (Button As Integer, X As Single, Y As Single)
   PaintNow = False   ' Turn off painting.
End Sub

Private Sub Form_MouseMove (Button As Integer, Shift As Integer, X As Single, Y As Single)
   If PaintNow Then
      PSet (X, Y)      ' Draw a point.
   End If
End Sub

Private Sub Form_Load ()
   DrawWidth = 10      ' Use wider brush.
   ForeColor = RGB(0, 0, 255)   ' Set drawing color.
End Sub
```

Visual Basic: Windows Controls

Visual Studio 6.0

NodeClick Event

See Also Example Applies To

Occurs when a **Node** object is clicked.

Syntax

```
Private Sub object_NodeClick(ByVal node As Node)
```

The NodeClick event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
node	A reference to the clicked **Node** object.

Remarks

The standard Click event is generated when the user clicks any part of the **TreeView** control outside a node object. The NodeClick event is generated when the user clicks a particular **Node** object; the NodeClick event also returns a reference to a particular **Node** object which can be used to validate the **Node** before further action is taken.

The NodeClick event occurs before the standard Click event.

Visual Basic: Windows Controls

NodeClick Event Example

This example adds several **Node** objects to a **TreeView** control. When a **Node** is clicked, the NodeClick event is triggered and is used to get the **Node** object's index and text. To try the example, place a **TreeView** control on a form and paste the code into the form's Declarations section. Run the example, and click any **Node**.

```
Private Sub Form_Load()
   Dim nodX As Node
   Set nodX = TreeView1.Nodes.Add(,,"R","Root")
   nodX.Expanded = True
   Set nodX = TreeView1.Nodes.Add(,,"P","Parent")
   nodX.Expanded = True
   Set nodX = TreeView1.Nodes.Add("R",tvwChild,,"Child 1")
   Set nodX = TreeView1.Nodes.Add("R",tvwChild,,"Child 2")
   Set nodX = TreeView1.Nodes.Add("R",tvwChild,,"Child 3")
   Set nodX = TreeView1.Nodes.Add("P",tvwChild,,"Child 4")
   Set nodX = TreeView1.Nodes.Add("P",tvwChild,,"Child 5")
   Set nodX = TreeView1.Nodes.Add("P",tvwChild,,"Child 6")
End Sub

Private Sub TreeView1_NodeClick(ByVal Node As Node)
   Form1.Caption = "Index = " & Node.Index & " Text:" & Node.Text
End Sub
```

Visual Basic: Windows Controls

Visual Studio 6.0

NodeCheck Event

See Also Example Applies To

Occurs when the **CheckBoxes** property equals **True** and a **Node** object is checked or unchecked.

Syntax

Event **NodeCheck**(**ByVal** *Node* **As ComctlLib.Node**)

The NodeCheck event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
Node	Returns a reference to the checked **Node** object.

Remarks

The event wil not occur when the **Node** object's **Checked** property is programmatically set to **True** or **False**.

The **CheckBoxes** property for the **TreeView** control must be set to **True** to display check boxes.