Visual Basic Reference

Visual Studio 6.0

ObjectEvent Event

See Also Example Applies To

Occurs when a control that is assigned to a **VBControlExtender** object variable raises an event.

Syntax

Private Sub *object*_**ObjectEvent**(*Info* **As EventInfo**)

The ObjectEvent event syntax has these parts:

Part	Description
object	Required. An object expression that evaluates to an object in the Applies To list.
Info	Returns a reference to an **EventInfo** object.

Remarks

The ObjectEvent event is a generic event that allows you to handle events of a control and return values through the control's event parameters.

You can use the ObjectEvent event to trap generic events a control raises, assuring that any control you deploy contains certain basic functionality required by the deployed application.

Visual Basic Reference

ObjectEvent Event, EventParameter Object Examples

The first example below uses the ObjectEvent event to print all parameter names and values.

```
Option Explicit
Private WithEvents extObj As VBControlExtender

' Code to set the object variable to a user control not shown.

Private Sub extObj_ObjectEvent(Info As EventInfo)
   Dim p As EventParameter
   Debug.Print Info.Name

   For Each p In Info.EventParameters
      Debug.Print p.Name, p.Value
   Next
End Sub
```

The second example allows you to check for a generic event raised by the control.

```
Private Sub extObj_ObjectEvent(Info As EventInfo)
   Dim p As EventParameter
   Select Case Info.Name
   Case "UserName"
      ' Check User name value.
      MsgBox Info.EventParameters("UserName").Value
   ' Other cases now shown
   Case Else ' Unknown Event
      ' Handle unknown events here.
   End Select
End Sub
```

Visual Basic Reference

Visual Studio 6.0

ObjectMove Event

See Also Example Applies To

Occurs immediately after the object within an **OLE** container control is moved or resized while the object is active.

Syntax

Private Sub *object*_**ObjectMove(***left* **As Single**, *top* **As Single**, *width* **As Single**, *height* **As Single)**

The ObjectMove event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
left	The coordinate of the left edge of the **OLE** container control immediately after its moved or resized.
top	The coordinate of the top edge of the **OLE** container control immediately after its moved or resized.
width	The width of the **OLE** container control immediately after it's moved or resized.
height	The height of the **OLE** container control immediately after it's moved or resized.

Remarks

When a user moves or resizes an **OLE** container control, your application can use the ObjectMove event to determine whether to actually change the size and position of the control. If the ObjectMove event procedure doesn't change the **OLE** container control's position or size, the object within the **OLE** container control returns to its original position and is informed of its new size. The coordinates passed as arguments to this event include the border of the **OLE** container control.

The ObjectMove and Resize events are triggered when the **OLE** container control receives information about the size of the object it contains. However, the Resize event doesn't receive any information about the position of the control. If the **OLE** container control is moved off the form, the arguments have negative or positive values that represent the position of the object relative to the top and left of the form.

Visual Basic Reference

Visual Studio 6.0

OLECompleteDrag Event (ActiveX Controls)

See Also Example Applies To

Occurs when a source component is dropped onto a target component, informing the source component that a drag action was either performed or canceled.

Syntax

Private Sub *object*_**OLECompleteDrag**([*effect* **As Long**])

The OLECompleteDrag event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
effect	A long integer set by the source object identifying the action that has been performed, thus allowing the source to take appropriate action if the component was moved (such as the source deleting data if it is moved from one component to another). The possible values are listed in Settings.

Settings

The settings for *effect* are:

Constant	Value	Description
vbDropEffectNone	0	Drop target cannot accept the data, or the drop operation was canceled.
vbDropEffectCopy	1	Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
vbDropEffectMove	2	Drop results in data being moved from the drag source to the drop source. The drag source should remove the data from itself after the move.

Remarks

The OLECompleteDrag event is the final event to be called in an OLE drag/drop operation. This event informs the source component of the action that was performed when the object was dropped onto the target component. The target sets this value through the *effect* parameter of the OLEDragDrop event. Based on this, the source can then determine the appropriate

action it needs to take. For example, if the object was moved into the target (**vbDropEffectMove**), the source needs to delete the object from itself after the move.

If **OLEDragMode** is set to **Automatic**, then Visual Basic handles the default behavior. The event still occurs, however, allowing the user to add to or change the behavior.

Most components support manual OLE drag and drop events, and some support automatic OLE drag and drop events.

Visual Basic Reference

Visual Studio 6.0

OLECompleteDrag Event

See Also Example Applies To

Occurs when a source component is dropped onto a target component, informing the source component that a drag action was either performed or canceled.

Syntax

Private Sub *object*_**OLECompleteDrag(**[*effect* **As Long**]**)**

The OLECompleteDrag event syntax has these parts:

Part	Description
Object	An object expression that evaluates to an object in the Applies To list.
Effect	A long integer set by the source object identifying the action that has been performed, thus allowing the source to take appropriate action if the component was moved (such as the source deleting data if it is moved from one component to another). The possible values are listed in Settings.

Settings

The settings for *effect* are:

Constant	Value	Description
VbDropEffectNone	0	Drop target cannot accept the data, or the drop operation was canceled.
VbDropEffectCopy	1	Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
VbDropEffectMove	2	Drop results in data being moved from the drag source to the drop source. The drag source should remove the data from itself after the move.

Remarks

The OLECompleteDrag event is the final event to be called in an OLE drag/drop operation. This event informs the source component of the action that was performed when the object was dropped onto the target component. The target sets this value through the *effect* parameter of the OLEDragDrop event. Based on this, the source can then determine the appropriate

action it needs to take. For example, if the object was moved into the target (**vbDropEffectMove**), the source needs to delete the object from itself after the move.

If **OLEDragMode** is set to **Automatic**, then Visual Basic handles the default behavior. The event still occurs, however, allowing the user to add to or change the behavior.

Most components support manual OLE drag and drop events, and some support automatic OLE drag and drop events.

Visual Basic Reference

Visual Studio 6.0

OLEDragDrop Event (ActiveX Controls)

See Also Example Applies To

Occurs when a source component is dropped onto a target component when the source component determines that a drop can occur.

Note This event occurs only if **OLEDropMode** is set to **1** (**Manual**).

Syntax

Private Sub *object*_**OLEDragDrop**(*data* **As DataObject**, *effect* **As Long**, *button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single**)

The OLEDragDrop event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
data	A **DataObject** object containing formats that the source will provide and, in addition, possibly the data for those formats. If no data is contained in the **DataObject**, it is provided when the control calls the **GetData** method. The **SetData** and **Clear** methods cannot be used here.
effect	A long integer set by the target component identifying the action that has been performed (if any), thus allowing the source to take appropriate action if the component was moved (such as the source deleting the data). The possible values are listed in Settings.
button	An integer which acts as a bit field corresponding to the state of a mouse button when it is depressed. The left button is bit 0, the right button is bit 1, and the middle button is bit 2. These bits correspond to the values 1, 2, and 4, respectively. It indicates the 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 depressed.
shift	An integer which acts as a bit field corresponding to the state of the shift, ctrl, and alt keys when they are depressed. The shift key is bit 0, the ctrl key is bit 1, and the alt key is bit 2. These bits correspond to the values 1, 2, and 4, respectively. The *shift* parameter 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 depressed. For example, if both the ctrl and alt keys were depressed, the value of *shift* would be 6.
x,y	A number which 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.

Settings

The settings for *effect* are:

Constant	Value	Description
vbDropEffectNone	0	Drop target cannot accept the data.
vbDropEffectCopy	1	Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
vbDropEffectMove	2	Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.
VbDropEffectScroll	-2147483648	A mask to indicate that the drop target window has scrolled/would scroll.

Remarks

The source ActiveX component should always mask values from the *effect* parameter to ensure compatibility with future implementations of ActiveX components. Presently, only three of the 32 bits in the *effect* parameter are used. In future versions of Visual Basic, however, these other bits may be used. Therefore, as a precaution against future problems, drag sources and drop targets should mask these values appropriately before performing any comparisons.

For example, a source component should not compare an *effect* against, for example, **vbDropEffectCopy**, such as:

```
If Effect = vbDropEffectCopy...
```

Instead, the source component should mask the value or values being sought, as here:

```
If Effect And vbDropEffectCopy = vbDropEffectCopy...
```

-or-

```
If (Effect And vbDropEffectCopy)...
```

This allows for the definition of new drop effects in future versions of Visual Basic while preserving backwards compatibility with your existing code.

Most components support manual OLE drag and drop events, and some support automatic OLE drag and drop events.

Visual Basic Reference

Visual Studio 6.0

OLEDragDrop Event

See Also Example Applies To

Occurs when a source component is dropped onto a target component when the source component determines that a drop can occur.

Note This event occurs only if **OLEDropMode** is set to **1** (**Manual**).

Syntax

Private Sub *object*_**OLEDragDrop(***data* **As DataObject**, *effect* **As Long**, *button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single)**

The OLEDragDrop event syntax has these parts:

Part	Description
Object	An object expression that evaluates to an object in the Applies To list.
Data	A **DataObject** object containing formats that the source will provide and, in addition, possibly the data for those formats. If no data is contained in the **DataObject**, it is provided when the control calls the **GetData** method. The **SetData** and **Clear** methods cannot be used here.
Effect	A long integer set by the target component identifying the action that has been performed (if any), thus allowing the source to take appropriate action if the component was moved (such as the source deleting the data). The possible values are listed in Settings.
Button	An integer which acts as a bit field corresponding to the state of a mouse button when it is depressed. The left button is bit 0, the right button is bit 1, and the middle button is bit 2. These bits correspond to the values 1, 2, and 4, respectively. It indicates the 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 depressed.
Shift	An integer which acts as a bit field corresponding to the state of the SHIFT, CTRL, and ALT keys when they are depressed. The SHIFT key is bit 0, the CTRL key is bit 1, and the ALT key is bit 2. These bits correspond to the values 1, 2, and 4, respectively. The *shift* parameter 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 depressed. For example, if both the CTRL and ALT keys were depressed, the value of *shift* would be 6.
x,y	A number which 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.

Settings

The settings for *effect* are:

Constant	Value	Description
VbDropEffectNone	0	Drop target cannot accept the data.
VbDropEffectCopy	1	Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
VbDropEffectMove	2	Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.
VbDropEffectScroll	-2147483648	A mask to indicate that the drop target window has scrolled or would scroll.

Remarks

The source ActiveX component should always mask values from the *effect* parameter to ensure compatibility with future implementations of ActiveX components. Presently, only three of the 32 bits in the *effect* parameter are used. In future versions of Visual Basic, however, these other bits may be used. Therefore, as a precaution against future problems, drag sources and drop targets should mask these values appropriately before performing any comparisons.

For example, a source component should not compare an *effect* against, say, **vbDropEffectCopy**, such as in this manner:

```
If Effect = vbDropEffectCopy...
```

Instead, the source component should mask for the value or values being sought, such as this:

```
If Effect And vbDropEffectCopy = vbDropEffectCopy...
```

-or-

```
If (Effect And vbDropEffectCopy)...
```

This allows for the definition of new drop effects in future versions of Visual Basic while preserving backwards compatibility with your existing code.

Most components support manual OLE drag and drop events, and some support automatic OLE drag and drop events.

Visual Basic Reference

Visual Studio 6.0

OLEDragOver Event (ActiveX Controls)

See Also Example Applies To

Occurs when one component is dragged over another.

Syntax

Private Sub *object*_**OLEDragOver**(*data* **As DataObject**, *effect* **As Long**, *button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single**, *state* **As Integer**)

The OLEDragOver event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
data	A **DataObject** object containing formats that the source will provide and, in addition, possibly the data for those formats. If no data is contained in the **DataObject**, it is provided when the control calls the **GetData** method. The **SetData** and **Clear** methods cannot be used here.
effect	A long integer initially set by the source object identifying all effects it supports. This parameter must be correctly set by the target component during this event. The value of *effect* is determined by logically **Or**'ing together all active effects (as listed in Settings). The target component should check these effects and other parameters to determine which actions are appropriate for it, and then set this parameter to one of the allowable effects (as specified by the source) to specify which actions will be performed if the user drops the selection on the component. The possible values are listed in Settings.
button	An integer which acts as a bit field corresponding to the state of a mouse button when it is depressed. The left button is bit 0, the right button is bit 1, and the middle button is bit 2. These bits correspond to the values 1, 2, and 4, respectively. It indicates the 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 depressed.
shift	An integer which acts as a bit field corresponding to the state of the shift, ctrl, and alt keys when they are depressed. The shift key is bit 0, the ctrl key is bit 1, and the alt key is bit 2. These bits correspond to the values 1, 2, and 4, respectively. The *shift* parameter 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 depressed. For example, if both the ctrl and alt keys are depressed, the value of *shift* would be 6.
x,y	A number that specifies the current horizontal (*x*) and vertical (*y*) position of the mouse pointer within the target form or control. 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.
state	An integer that corresponds to the transition state of the control being dragged in relation to a target form or control. The possible values are listed in Settings.

Settings

The settings for *effect* are:

Constant	Value	Description
vbDropEffectNone	0	Drop target cannot accept the data.
vbDropEffectCopy	1	Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
vbDropEffectMove	2	Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.
vbDropEffectScroll	-2147483648 (&H80000000)	Scrolling is occurring or about to occur in the target component. This value is used in conjunction with the other values. **Note** Use only if you are performing your own scrolling in the target component.

The settings for *state* are:

Constant	Value	Description
vbEnter	0	Source component is being dragged within the range of a target.
vbLeave	1	Source component is being dragged out of the range of a target.
vbOver	2	Source component has moved from one position in the target to another.

Remarks

Note If the *state* parameter is **vbLeave**, indicating that the mouse pointer has left the target, then the *x* and *y* parameters will contain zeros.

The source component should always mask values from the *effect* parameter to ensure compatibility with future implementations of ActiveX components. Presently, only three of the 32 bits in the *effect* parameter are used. In future versions of Visual Basic, however, these other bits may be used. Therefore, as a precaution against future problems, drag sources and drop targets should mask these values appropriately before performing any comparisons.

For example, a source component should not compare an *effect* against, for example, **vbDropEffectCopy**, such as:

```
If Effect = vbDropEffectCopy...
```

Instead, the source component should mask for the value or values being sought, as here:

```
If Effect And vbDropEffectCopy = vbDropEffectCopy...
```

-or-

```
If (Effect And vbDropEffectCopy)...
```

This allows for the definition of new drop effects in future versions of Visual Basic while preserving backwards compatibility with your existing code.

Most components support manual OLE drag and drop events, and some support automatic OLE drag and drop events.

Visual Basic Reference

Visual Studio 6.0

OLEDragOver Event

See Also Example Applies To

Occurs when one component is dragged over another.

Syntax

Private Sub *object*_**OLEDragOver(***data* **As DataObject**, *effect* **As Long**, *button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single**, *state* **As Integer)**

The OLEDragOver event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
data	A **DataObject** object containing formats that the source will provide and, in addition, possibly the data for those formats. If no data is contained in the **DataObject**, it is provided when the control calls the **GetData** method. The **SetData** and **Clear** methods cannot be used here.
effect	A long integer initially set by the source object identifying all effects it supports. This parameter must be correctly set by the target component during this event. The value of *effect* is determined by logically **Or**'ing together all active effects (as listed in Settings). The target component should check these effects and other parameters to determine which actions are appropriate for it, and then set this parameter to one of the allowable effects (as specified by the source) to specify which actions will be performed if the user drops the selection on the component. The possible values are listed in Settings.
button	An integer which acts as a bit field corresponding to the state of a mouse button when it is depressed. The left button is bit 0, the right button is bit 1, and the middle button is bit 2. These bits correspond to the values 1, 2, and 4, respectively. It indicates the 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 depressed.
shift	An integer which acts as a bit field corresponding to the state of the SHIFT, CTRL, and ALT keys when they are depressed. The SHIFT key is bit 0, the CTRL key is bit 1, and the ALT key is bit 2. These bits correspond to the values 1, 2, and 4, respectively. The *shift* parameter 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 depressed. For example, if both the CTRL and ALT keys are depressed, the value of *shift* would be 6.
x,y	A number that specifies the current horizontal (*x*) and vertical (*y*) position of the mouse pointer within the target form or control. 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.
state	An integer that corresponds to the transition state of the control being dragged in relation to a target form or control. The possible values are listed in Settings.

Settings

The settings for *effect* are:

Constant	Value	Description
vbDropEffectNone	0	Drop target cannot accept the data.
vbDropEffectCopy	1	Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
vbDropEffectMove	2	Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.
vbDropEffectScroll	-2147483648 (&H80000000)	Scrolling is occurring or about to occur in the target component. This value is used in conjunction with the other values. **Note** Use only if you are performing your own scrolling in the target component.

The settings for *state* are:

Constant	Value	Description
vbEnter	0	Source component is being dragged within the range of a target.
vbLeave	1	Source component is being dragged out of the range of a target.
vbOver	2	Source component has moved from one position in the target to another.

Remarks

Note If the *state* parameter is **vbLeave**, indicating that the mouse pointer has left the target, then the *x* and *y* parameters will contain zeros.

The source component should always mask values from the *effect* parameter to ensure compatibility with future implementations of ActiveX components. Presently, only three of the 32 bits in the *effect* parameter are used. In future versions of Visual Basic, however, these other bits may be used. Therefore, as a precaution against future problems, drag sources and drop targets should mask these values appropriately before performing any comparisons.

For example, a source component should not compare an *effect* against, say, **vbDropEffectCopy**, such as in this manner:

```
If Effect = vbDropEffectCopy...
```

Instead, the source component should mask for the value or values being sought, such as this:

```
If Effect And vbDropEffectCopy = vbDropEffectCopy...
```

-or-

```
If (Effect And vbDropEffectCopy)...
```

This allows for the definition of new drop effects in future versions of Visual Basic while preserving backwards compatibility with your existing code.

Most components support manual OLE drag and drop events, and some support automatic OLE drag and drop events.

Visual Basic Reference

Visual Studio 6.0

OLEGiveFeedback Event (ActiveX Controls)

See Also Example Applies To

Occurs after every OLEDragOver event. OLEGiveFeedback allows the source component to provide visual feedback to the user, such as changing the mouse cursor to indicate what will happen if the user drops the object, or provide visual feedback on the selection (in the source component) to indicate what will happen.

Syntax

Private Sub *object*_**OLEGiveFeedback**(*effect* **As Long**, *defaultcursors* **As Boolean**)

The OLEGiveFeedback event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
effect	A long integer set by the target component in the OLEDragOver event specifying the action to be performed if the user drops the selection on it. This allows the source to take the appropriate action (such as giving visual feedback). The possible values are listed in Settings.
defaultcursors	A boolean value which determines whether Visual Basic uses the default mouse cursor provided by the component, or uses a user-defined mouse cursor.
	True (default) = use default mouse cursor.
	False = do not use default cursor. Mouse cursor must be set with the **MousePointer** property of the **Screen** object.

Settings

The settings for *effect* are:

Constant	Value	Description
vbDropEffectNone	0	Drop target cannot accept the data.
vbDropEffectCopy	1	Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.

vbDropEffectMove	2	Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.
vbDropEffectScroll	-2147483648 (&H80000000)	Scrolling is occurring or about to occur in the target component. This value is used in conjunction with the other values. **Note** Use only if you are performing your own scrolling in the target component.

Remarks

If there is no code in the OLEGiveFeedback event, or if the *defaultcursors* parameter is set to **True**, then Visual Basic automatically sets the mouse cursor to the default cursor provided by the component.

The source component should always mask values from the *effect* parameter to ensure compatibility with future implementations of components. Presently, only three of the 32 bits in the *effect* parameter are used. In future versions of Visual Basic, however, these other bits may be used. Therefore, as a precaution against future problems, drag sources and drop targets should mask these values appropriately before performing any comparisons.

For example, a source component should not compare an *effect* against, for example, **vbDropEffectCopy**, such as:

```
If Effect = vbDropEffectCopy...
```

Instead, the source component should mask for the value or values being sought, as here:

```
If Effect And vbDropEffectCopy = vbDropEffectCopy...
```

-or-

```
If (Effect And vbDropEffectCopy)...
```

This allows for the definition of new drop effects in future versions of Visual Basic while preserving backwards compatibility with your existing code.

Most components support manual OLE drag and drop events, and some support automatic OLE drag and drop events.

Visual Basic Reference

Visual Studio 6.0

OLEGiveFeedback Event

See Also Example Applies To

Occurs after every OLEDragOver event. OLEGiveFeedback allows the source component to provide visual feedback to the user, such as changing the mouse cursor to indicate what will happen if the user drops the object, or provide visual feedback on the selection (in the source component) to indicate what will happen.

Syntax

Private Sub *object*_**OLEGiveFeedback(***effect* **As Long**, defaultcursors **As Boolean)**

The OLEGiveFeedback event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
effect	A long integer set by the target component in the OLEDragOver event specifying the action to be performed if the user drops the selection on it. This allows the source to take the appropriate action (such as giving visual feedback). The possible values are listed in Settings.
defaultcursors	A boolean value which determines whether Visual Basic uses the default mouse cursor proved by the component, or uses a user-defined mouse cursor. **True** (default) = use default mouse cursor. **False** = do not use default cursor. Mouse cursor must be set with the **MousePointer** property of the **Screen** object.

Settings

The settings for *effect* are:

Constant	Value	Description
vbDropEffectNone	0	Drop target cannot accept the data.
vbDropEffectCopy	1	Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
vbDropEffectMove	2	Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.

vbDropEffectScroll	-2147483648 (&H80000000)	Scrolling is occurring or about to occur in the target component. This value is used in conjunction with the other values. **Note** Use only if you are performing your own scrolling in the target component.

Remarks

If there is no code in the OLEGiveFeedback event, or if the *defaultcursors* parameter is set to **True**, then Visual Basic automatically sets the mouse cursor to the default cursor provided by the component.

The source component should always mask values from the *effect* parameter to ensure compatibility with future implementations of components. Presently, only three of the 32 bits in the *effect* parameter are used. In future versions of Visual Basic, however, these other bits may be used. Therefore, as a precaution against future problems, drag sources and drop targets should mask these values appropriately before performing any comparisons.

For example, a source component should not compare an *effect* against, say, **vbDropEffectCopy**, such as in this manner:

```
If Effect = vbDropEffectCopy...
```

Instead, the source component should mask for the value or values being sought, such as this:

```
If Effect And vbDropEffectCopy = vbDropEffectCopy...
```

-or-

```
If (Effect And vbDropEffectCopy)...
```

This allows for the definition of new drop effects in future versions of Visual Basic while preserving backwards compatibility with your existing code.

Most components support manual OLE drag and drop events, and some support automatic OLE drag and drop events.

Visual Basic Reference

Visual Studio 6.0

OLESetData Event (ActiveX Controls)

See Also Example Applies To

Occurs on a source component when a target component performs the **GetData** method on the sources **DataObject** object, but before the data for the specified format has been loaded.

Syntax

Private Sub *object*_**OLESetData**(*data* **As DataObject**, *dataformat* **As Integer**)

The OLESetData event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
data	A **DataObject** object in which to place the requested data. The component calls the **SetData** method to load the requested format.
dataformat	An integer specifying the format of the data that the target component is requesting. The source component uses this value to determine what to load into the **DataObject** object.

Remarks

In certain cases, you may want to defer loading data into the **DataObject** object of a source component to save time, especially if the source component supports many formats. This event allows the source to respond to only one request for a given format of data. When this event is called, the source should check the *format* parameter to determine what needs to be loaded and then perform the **SetData** method on the **DataObject** object to load the data which is then passed back to the target component.

Visual Basic Reference

Visual Studio 6.0

OLESetData Event

See Also Example Applies To

Occurs on an source component when a target component performs the **GetData** method on the sources **DataObject** object, but the data for the specified format has not yet been loaded.

Syntax

Private Sub *object*_**OLESetData(***data* **As DataObject**, *dataformat* **As Integer)**

The OLESetData event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
data	A **DataObject** object in which to place the requested data. The component calls the **SetData** method to load the requested format.
dataformat	An integer specifying the format of the data that the target component is requesting. The source component uses this value to determine what to load into the **DataObject** object.

Remarks

In certain cases, you may wish to defer loading data into the **DataObject** object of a source component to save time, especially if the source component supports many formats. This event allows the source to respond to only one request for a given format of data. When this event is called, the source should check the *format* parameter to determine what needs to be loaded and then perform the **SetData** method on the **DataObject** object to load the data which is then passed back to the target component.

Visual Basic Reference

Visual Studio 6.0

OLEStartDrag Event (ActiveX Controls)

See Also Example Applies To

Occurs when a component's **OLEDrag** method is performed, or when a component initiates an OLE drag/drop operation when the **OLEDragMode** property is set to **Automatic**.

This event specifies the data formats and drop effects that the source component supports. It can also be used to insert data into the **DataObject** object.

Syntax

Private Sub *object*_**OLEStartDrag**(*data* **As DataObject**, *allowedeffects* **As Long**)

The OLEStartDrag event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
data	A **DataObject** object containing formats that the source will provide and, optionally, the data for those formats. If no data is contained in the **DataObject**, it is provided when the control calls the **GetData** method, and you should provide the values for the *data* parameter. The **SetData** and **Clear** methods cannot be used here.
allowedeffects	A long integer containing the effects that the source component supports. The possible values are listed in Settings.

Settings

The settings for *allowedeffects* are:

Constant	Value	Description
vbDropEffectNone	0	Drop target cannot accept the data.
vbDropEffectCopy	1	Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
vbDropEffectMove	2	Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.

Remarks

The source component should logically **Or** together the supported values and place the result in the *allowedeffects* parameter. The target component can use this value to determine the appropriate action, and what the appropriate user feedback should be.

The StartDrag event also occurs if the components **OLEDragMode** property is set to **Automatic**. This allows you to add formats and data to the **DataObject** object after the component has done so. You can also override the default behavior of the component by clearing the **DataObject** object (using the **Clear** method) and then adding your data and formats.

You may want to defer placing data into the **DataObject** object until the target component requests it. This allows the source component to save time by not loading multiple data formats. When the target performs the **GetData** method on the **DataObject**, the sources OLESetData event will occur if the requested data is not contained in the **DataObject**. At this point, the data can be loaded into the **DataObject**, which will in turn provide the data to the target.

If the user does not load any formats into the **DataObject**, then the drag/drop operation is canceled.

Visual Basic Reference

Visual Studio 6.0

OLEStartDrag Event

See Also Example Applies To

Occurs when a component's **OLEDrag** method is performed, or when a component initiates an OLE drag/drop operation when the **OLEDragMode** property is set to **Automatic**.

This event specifies the data formats and drop effects that the source component supports. It can also be used to insert data into the **DataObject** object.

Syntax

Private Sub *object*_**OLEStartDrag(***data* **As DataObject**, *allowedeffects* **As Long)**

The OLEStartDrag event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
data	A **DataObject** object containing formats that the source will provide and, optionally, the data for those formats. If no data is contained in the **DataObject**, it is provided when the control calls the **GetData** method. The programmer should provide the values for this parameter in this event. The **SetData** and **Clear** methods cannot be used here.
allowedeffects	A long integer containing the effects that the source component supports. The possible values are listed in Settings. The programmer should provide the values for this parameter in this event.

Settings

The settings for *allowedeffects* are:

Constant	Value	Description
vbDropEffectNone	0	Drop target cannot accept the data.
vbDropEffectCopy	1	Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
vbDropEffectMove	2	Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.

Remarks

The source component should use the logical **Or** operator against the supported values and place the result in the *allowedeffects* parameter. The target component can use this value to determine the appropriate action (and what the appropriate user feedback should be).

The OLEStartDrag event also occurs if the components **OLEDragMode** property is set to **Automatic**. This allows you to add formats and data to the **DataObject** object after the component has done so. You can also override the default behavior of the component by clearing the **DataObject** object (using the **Clear** method) and then adding your data and formats.

You may wish to defer putting data into the **DataObject** object until the target component requests it. This allows the source component to save time by not loading multiple data formats. When the target performs the **GetData** method on the **DataObject**, the sources OLESetData event will occur if the requested data is not contained in the **DataObject**. At this point, the data can be loaded into the **DataObject**, which will in turn provide the data to the target.

If the user does not load any formats into the **DataObject**, then the drag/drop operation is canceled.

Visual Basic: DataGrid Control

Visual Studio 6.0

OnAddNew Event

See Also Example Applies To

Occurs when a user action invokes an AddNew operation.

Syntax

Private Sub *object*_**OnAddNew(**[*index* **As Integer**]**)**

The OnAddNew 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 identifies a control if it is in a control array.

Remarks

The OnAddNew event occurs when an AddNew operation has been initiated by either of the following:

- The user modifies a cell within the AddNew row. Typically, this occurs as soon as the user types a character, but may also occur as a result of a built-in radio button or combo box selection.
- The **Value** or **Text** property of a column is set in code when the current cell is within the AddNew row.

This event occurs only if the grid's **AllowAddNew** property is **True**.

When the OnAddNew event is fired, the value of the **AddNewMode** property is 2 - AddNew Pending.

Visual Basic: MSComm Control

Visual Studio 6.0

OnComm Event

See Also Example Applies To

The **OnComm** event is generated whenever the value of the **CommEvent** property changes, indicating that either a communication event or an error occurred.

Syntax

Private Sub *object*_**OnComm ()**

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

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

Remarks

The **CommEvent** property contains the numeric code of the actual error or event that generated the **OnComm** event. Note that setting the **RThreshold** or **SThreshold** properties to 0 disables trapping for the **comEvReceive** and **comEvSend** events, respectively.

Visual Basic: MSComm Control

OnComm Event Example

The following example shows how to handle communications errors and events. You can insert code after each related Case statement, to handle a particular error or event.

```
Private Sub MSComm_OnComm ()
   Select Case MSComm1.CommEvent
   ' Handle each event or error by placing
   ' code below each case statement

   ' Errors
      Case comEventBreak   ' A Break was received.
      Case comEventFrame   ' Framing Error
      Case comEventOverrun   ' Data Lost.
      Case comEventRxOver   ' Receive buffer overflow.
      Case comEventRxParity   ' Parity Error.
      Case comEventTxFull   ' Transmit buffer full.
      Case comEventDCB   ' Unexpected error retrieving DCB]

   ' Events
      Case comEvCD   ' Change in the CD line.
      Case comEvCTS   ' Change in the CTS line.
      Case comEvDSR   ' Change in the DSR line.
      Case comEvRing   ' Change in the Ring Indicator.
      Case comEvReceive   ' Received RThreshold # of
                        ' chars.
      Case comEvSend   ' There are SThreshold number of
                     ' characters in the transmit
                     ' buffer.
      Case comEvEof   ' An EOF charater was found in
                     ' the input stream
   End Select
End Sub
```

Visual Basic Reference

Visual Studio 6.0

Paint Event

See Also Example Applies To

Occurs when part or all of an object is exposed after being moved or enlarged, or after a window that was covering the object has been moved.

Syntax

```
Private Sub Form_Paint( )

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

The Paint 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 Paint event procedure is useful if you have output from graphics methods in your code. With a Paint procedure, you can ensure that such output is repainted when necessary.

The Paint event is invoked when the **Refresh** method is used. If the **AutoRedraw** property is set to **True**, repainting or redrawing is automatic, so no Paint events are necessary.

If the **ClipControls** property is set to **False**, graphics methods in the Paint event procedure affect only newly exposed areas of the form; otherwise, the graphics methods repaint all areas of the form not covered by controls (except **Image**, **Label**, **Line**, and **Shape** controls).

Using a **Refresh** method in a Resize event procedure forces repainting of the entire object every time a user resizes the form.

Note Using a Paint event procedure for certain tasks can cause a cascading event. In general, avoid using a Paint event procedure to do the following:

- Move or size a form or control.
- Change any variables that affect size or appearance, such as setting an object's **BackColor** property.
- Invoke a **Refresh** method.

A Resize event procedure may be more appropriate for some of these tasks.

Visual Basic Reference

Paint Event Example

This example draws a diamond that intersects the midpoint of each side of a form and adjusts automatically as the form is resized. To try this example, paste the code into the Declarations section of a form, and then press F5 and resize the form.

```
Private Sub Form_Paint ()
   Dim HalfX, HalfY   ' Declare variables.
   HalfX = ScaleLeft + ScaleWidth / 2   ' Set to one-half of width.
   HalfY = ScaleTop + ScaleHeight / 2   ' Set to one-half of height.
   ' Draw a diamond.
   Line (ScaleLeft, HalfY) - (HalfX, ScaleTop)
   Line -(ScaleWidth + ScaleLeft, HalfY)
   Line -(HalfX, ScaleHeight + ScaleTop)
   Line -(ScaleLeft, HalfY)
End Sub

Private Sub Form_Resize
   Refresh
End Sub
```

Visual Basic: Windows Controls

Visual Studio 6.0

PanelClick Event

See Also Example Applies To

The PanelClick event is similar to the standard Click event but occurs when a user presses and then releases a mouse button over any of the **StatusBar** control's **Panel** objects.

Syntax

Private Sub *object*_**PanelClick(ByVal** *panel* **As Panel)**

The PanelClick event syntax has these parts:

Part	Description
object	An object expression that evaluates to a **StatusBar** control.
panel	A reference to a **Panel** object.

Remarks

The standard Click event also occurs when a **Panel** object is clicked.

The PanelClick event is only generated when the click occurs over a **Panel** object. When the **StatusBar** control's **Style** property is set to Simple style, panels are hidden, and therefore the PanelClick event is not generated.

You can use the reference to the **Panel** object to set properties for that panel. For example, the following code resets the **Bevel** property of a clicked **Panel**:

```
Private Sub StatusBar1_PanelClick(ByVal Panel As Panel)
   Select Case Panel.Key
   Case "DisplayFileName" ' Key="DisplayFileName"
     Panel.Bevel = sbrRaised    ' Reset Bevel property
   ' Add other case statements for other panels
   End Select
End Sub
```

Visual Basic: Windows Controls

PanelClick Event Example

This example adds two **Panel** objects to a **StatusBar** control; when each **Panel** is clicked, the value of the **Key** and **Width** properties of the clicked **Panel** are displayed in the third **Panel**. To try the example, place a **StatusBar** control on a form and paste the code into the Declarations section. Run the example.

```
Private Sub Form_Load()
   Dim I as Integer
   For I = 1 to 2
      StatusBar1.Panels.Add
   Next I

   With StatusBar1.Panels
      .Item(1).Style = sbrDate
      .Item(1).Key = "Date panel"
      .Item(1).AutoSize = sbrContents
      .Item(1).MinWidth = 2000
      .Item(2).Style = sbrTime
      .Item(2).Key = "Time panel"
      .Item(3).AutoSize = sbrContents   ' Content
      .Item(3).Text = "Panel 3"
      .Item(3).Key = "Panel 3"
   End With
End Sub

Private Sub StatusBar1_PanelClick(ByVal Panel As Panel)
   ' Show clicked panel's key and width in Panel 3.
   StatusBar1.Panels("Panel 3").Text = Panel.Key & " Width = " & Panel.Width
End Sub
```

Visual Basic: Windows Controls

Visual Studio 6.0

PanelDblClick Event

See Also Example Applies To

The PanelDblClick event is similar to the standard DblClick Event but occurs when a user presses and then releases a mouse button twice over a **StatusBar** control's **Panel** object.

Syntax

Sub *object*_**PanelDblClick(ByVal** *panel* **As Panel)**

The PanelDblClick event syntax has these parts:

Part	Description
object	An object expression that evaluates to a **StatusBar** control.
panel	A reference to the double-clicked **Panel**.

Remarks

The standard DblClick event also occurs when a **Panel** is double-clicked.

The PanelDblClick event is only generated when the double-click occurs over a **Panel** object. When the **StatusBar** control's **Style** property is set to Simple style, panels are hidden, and therefore the PanelDblClick event is not generated.

Visual Basic: Windows Controls

PanelDblClick Event Example

This example adds two **Panel** objects to a **StatusBar** control. When the user double-clicks on the control, the text of the clicked **Panel** object is displayed. To try the example, place a **StatusBar** control on a form and paste the code into the form's Declarations section. Run the example and double-click on the control.

```
Private Sub Form_Load()
Dim I as Integer
   For I = 1 to 2
      StatusBar1.Panels.Add
   Next I

   With StatusBar1.Panels
      .Item(1).Text = "A long piece of information."
      .Item(1).AutoSize = sbrContents   ' Content
      .Item(2).Style = sbrDate   ' Date style
      .Item(2).AutoSize = sbrContents   ' Content
      .Item(3).Style = sbrTime   ' Time style
   End With
End Sub

Private Sub StatusBar1_PanelDblClick(ByVal Panel As Panel)
   MsgBox "Panel.Style = " & Panel.Style
End Sub
```

Visual Basic Reference

Visual Studio 6.0

PathChange Event

See Also Example Applies To

Occurs when the path is changed by setting the **FileName** or **Path** property in code.

Syntax

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

The PathChange 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

You can use a PathChange event procedure to respond to path changes in a **FileListBox** control. When you assign a string containing a new path to the **FileName** property, the **FileListBox** control invokes the PathChange event.

Visual Basic Reference

PathChange Event Example

This example demonstrates how to update a **Label** control to reflect the current path for a **FileListBox** control. Double-clicking a directory name displays a list of that directory's files in the **FileListBox**; it also displays the directory's complete path in the **Label** control. To try this example, paste the code into the Declarations section of a form that contains a **Label** control, a **DirListBox** control, and a **FileListBox** control, and then press F5. Double-click a directory to change the path.

```
Private Sub File1_PathChange ()
   Label1.Caption = "Path: " & Dir1.Path   ' Show path in Label.
End Sub

Private Sub Dir1_Change ()
   File1.Path = Dir1.Path            ' Set file path.
End Sub

Private Sub Form_Load ()
   Label1.Caption = "Path: " & Dir1.Path   ' Show path in Label.
End Sub
```

Visual Basic Reference

Visual Studio 6.0

PatternChange Event

See Also Example Applies To

Occurs when the file listing pattern, such as "*.*", is changed by setting the **FileName** or **Pattern** property in code.

Syntax

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

The PatternChange 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

You can use a PatternChange event procedure to respond to pattern changes in a **FileListBox** control. When you assign a string containing a new pattern to the **FileName** property, the **FileListBox** invokes the PathChange event.

Visual Basic Reference

PatternChange Event Example

This example updates a **FileListBox** control with files matching the pattern entered in a **TextBox** control. If a full path is entered into the **TextBox**, such as C:\Bin*.exe, the text is automatically parsed into path and pattern components. To try this example, paste the code into the Declarations section of a form that contains a **TextBox** control, a **Label** control, a **FileListBox** control, and a **CommandButton** control, and then press F5 and enter a valid file pattern in the **TextBox**.

```
Private Sub Form_Load ()
   Command1.Default = True         ' Set Default property.
   Command1.Caption = "OK"         ' Set Caption.
End Sub

Private Sub Command1_Click ()   ' OK button clicked.
   ' Text is parsed into path and pattern components.
   File1.FileName = Text1.Text
   Label1.Caption = "Path: " & File1.Path
End Sub

Private Sub File1_PatternChange ()
   Text1.Text = File1.Pattern   ' Set text to new pattern.
End Sub
```

Visual Studio 6.0

Visual Basic: MSChart Control

PlotActivated Event

See Also Example Applies To

Occurs when the user double clicks the chart plot.

Syntax

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

The PlotActivated 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

PlotSelected Event

See Also Example Applies To

Occurs when the user clicks the chart plot.

Syntax

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

The PlotSelected 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

PlotUpdated Event

See Also Example Applies To

Occurs when the chart plot has changed.

Syntax

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

The PlotUpdated 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 plot, 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 Studio 6.0

Visual Basic: MSChart Control

PointActivated Event

See Also Example Applies To

Occurs when the user double clicks on a data point.

Syntax

Private Sub *object*_**PointActivated** (*series* **As Integer**, *dataPoint* **As Integer**, *mouseFlags* **As Integer**, *cancel* **As Integer**)

The PointActivated event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
series	Integer. Identifies the series containing the data point. Series are numbered in the order that their columns appear in the data grid, beginning with 1.
dataPoint	Integer. Identifies the data point's position in the series. Points are numbered in the order that their rows appear in the data grid, beginning with 1.
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 Basic: MSChart Control

PointLabelActivated Event

See Also Example Applies To

Occurs when the user double clicks a data point label.

Syntax

Private Sub *object*_**PointLabelActivated** (*series* **As Integer**, *dataPoint* **As Integer**, *mouseFlags* **As Integer**, *cancel* **As Integer**)

The PointLabelActivated event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
series	Integer. Identifies the series containing the data point. Series are numbered in the order that their columns appear in the data grid, beginning with 1.
dataPoint	Integer. Identifies the data point's position in the series. Points are numbered in the order that their rows appear in the data grid, beginning with 1.
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

PointLabelSelected Event

See Also Example Applies To

Occurs when the user clicks a data point label.

Syntax

Private Sub *object*_**PointLabelSelected** (*series* **As Integer**, *dataPoint* **As Integer**, *mouseFlags* **As Integer**, *cancel* **As Integer**)

The PointLabelSelected event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
series	Integer. Identifies the series containing the data point. Series are numbered in the order that their columns appear in the data grid, beginning with 1.
dataPoint	Integer. Identifies the data point's position in the series. Points are numbered in the order that their rows appear in the data grid, beginning with 1.
mouseFlags	Integer. Indicates whether a key is held down when the mouse button is clicked, as described in Settings.
cancel	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

PointLabelUpdated Event

See Also Example Applies To

Occurs when a data point label has changed.

Syntax

Private Sub *object*_**PointLabelUpdated** (*series* **As Integer**, *dataPoint* **As Integer**, *updateFlags* **As Integer**)

The PointLabelUpdated event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
series	Integer. Identifies the series containing the data point. Series are numbered in the order that their columns appear in the data grid, beginning with 1.
dataPoint	Integer. Identifies the data point's position in the series. Points are numbered in the order that their rows appear in the data grid, beginning with 1.
updateFlags	Integer. Provides information about the update of the data point label, 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 Studio 6.0

Visual Basic: MSChart Control

PointSelected Event

See Also Example Applies To

Occurs when the user clicks a data point.

Syntax

Private Sub *object*_**PointSelected** (*series* **As Integer**, *dataPoint* **As Integer**, *mouseFlags* **As Integer**, *cancel* **As Integer**)

The PointSelected event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
series	Integer. Identifies the series containing the data point. Series are numbered in the order that their columns appear in the data grid, beginning with 1.
dataPoint	Integer. Identifies the data point's position in the series. Points are numbered in the order that their rows appear in the data grid, beginning with 1.
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

PointUpdated Event

See Also Example Applies To

Occurs when a data point has changed.

Syntax

Private Sub *object*_**PointUpdated** (*series* **As Integer**, *dataPoint* **As Integer**, *updateFlags* **As Integer**)

The PointUpdated event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
series	Integer. Identifies the series containing the data point. Series are numbered in the order that their columns appear in the data grid, beginning with 1.
dataPoint	Integer. Identifies the data point's position in the series. Points are numbered in the order that their rows appear in the data grid, beginning with 1.
updateFlags	Integer. Provides information about the update of the data point, 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.

This documentation is archived and is not being maintained.

Visual Basic: SysInfo Control

Visual Studio 6.0

PowerQuerySuspend Event

See Also Example Applies To

Occurs when system power is about to be suspended.

Syntax

Private Sub *object*_**PowerQuerySuspend(**[*index* **As Integer,**] *cancel* **As Boolean)**

The PowerQuerySuspend event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	A numeric expression that evaluates to the index of a control if it is in a control array.
cancel	A Boolean value that indicates whether or not the system will be prevented from entering suspended mode, as described in Settings.

Settings

The settings for *cancel* are:

Setting	Description
True	Prevents the system from entering suspended mode.
False	(Default) Allows the system to enter suspended mode.

Visual Basic: SysInfo Control

Visual Studio 6.0

PowerResume Event

See Also Example Applies To

Occurs when the system comes out of suspend mode and applications resume normal operations.

Syntax

Private Sub *object*_**PowerResume(**[*index* **As Integer**]**)**

The PowerResume event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	A numeric expression that evaluates to the index of a control if it is in a control array.

This documentation is archived and is not being maintained.

Visual Basic: SysInfo Control

Visual Studio 6.0

PowerStatusChanged Event

See Also Example Applies To

Occurs when there is a change in the power status of the system.

Syntax

Private Sub *object*_**PowerStatusChanged(**[*index* **As Integer**]**)**

The PowerStatusChanged event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	A numeric expression that evaluates to the index of a control if it is in a control array.

Remarks

The following changes in the system's power status triggers the PowerStatusChanged event:

- Battery running low
- Switching power from AC to battery and vice versa
- Battery has completed charging

Visual Basic: SysInfo Control

PowerStatusChanged Event Example

This example updates a **Label** control each time there is a change in power status so the application displays current battery status information. To run this example, put a **SysInfo** control and a **Label** control on a form. Paste this code into the PowerStatusChanged event of the **SysInfo** control. Then run the example.

```
Private Sub SysInfo1_PowerStatusChanged()
   Select Case SysInfo1.BatteryStatus
      Case 1
         Label1.Caption = "Battery OK"
      Case 2
         Label1.Caption = "Battery Low"
      Case 4
         Label1.Caption = "Battery Critical"
      Case 8
         Label1.Caption = "Battery Charging"
      Case 128, 255
         Label1.Caption = "No Battery Status"
   End Select
End Sub
```

Visual Basic: SysInfo Control

Visual Studio 6.0

PowerSuspend Event

See Also Example Applies To

Occurs immediately before the system goes into suspend mode.

Syntax

Private Sub *object*_**PowerSuspend(**[*index* **As Integer**]**)**

The PowerSuspend event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	A numeric expression that evaluates to the index of a control if it is in a control array.

Visual Basic Reference

Visual Studio 6.0

ProcessingTimeout Event

See Also Example Applies To

Occurs at preset intervals, allowing the user to cancel an asynchronous operation.

Syntax

Private Sub *object*_**ProcessingTimeout(***Seconds* **As Long**, *Cancel* **As Boolean**, *JobType* **As AsyncTypeConstants**, *Cookie* **As Long)**

The ProcessingTimeout event syntax has these parts:

Part	Description
object	Required. An object expression that evaluates to an object in the Applies To list.
Seconds	Returns the number of seconds that have passed since starting the asynchronous operation.
Cancel	Sets a value that determines if the operation will be canceled, as shown in Settings.
JobType	Returns the type of operation, as shown in Settings.
Cookie	Returns the ID of the operation. The ID is set when an asynchronous method such as **ExportReport** or **PrintReport** is invoked.

Settings

The *JobType* settings are:

Constant	Value	Description
rptAsyncPreview	0	The report is processing a Preview operation.
rptAsyncPrint	1	The report is processing a Print operation.
rptAsyncReport	2	The report is processing an ExportReport operation.

The settings for *cancel* are:

Setting	Description
True	The operation is cancelled.
False	(Default) The operation continues.

Remarks

Use the ProcessingTimeout event to allow the user to cancel an asynchronous operation at specified periods of time.

Visual Basic Reference

Visual Studio 6.0

ProcessTag Event

See Also Example Applies To

When a token-prefixed tag is found in an HTML template during **WriteTemplate** processing, this event is fired to allow the **WebClass** to replace the contents of that tag.

Syntax

Private Sub *object*_**ProcessTag(***TagName* **As String,** *TagContents* **As String**, *SendTag* **As Boolean)**

The ProcessTag event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
TagName	The name of the tag being replaced.
TagContents	When the event is fired, contains the current contents of the tag. The WebClass code may change this variable to replace the contents appropriately.
SendTag	When set to **True**, both the tags and the contents are replaced in the specified replacement area. When set to **False**, only the value of the *TagContents* parameter will be sent to the file, to replace the entire replacement area. Default value is **False**.

Remarks

Event only fires when processing a template that contains replacement tags.

Visual Basic Reference

Visual Studio 6.0

QueryClose Event

See Also Example Applies To

Occurs before a form is terminated.

Syntax

Public Sub *object*_**QueryClose(***cancel* **As Integer**, *closemode* **As Integer)**

The QueryClose event syntax has these parts:

Part	Description
object	Required. An object expression that evaluates to an object in the Applies To list.
cancel	Set to any value other than zero to stop the designer from closing.
closemode	A value or constant indicating the cause of the QueryClose event, as described in Return Values.

Return Values

The *closemode* argument returns the following values:

Constant	Value	Description
vbFormControlMenu	0	The user has chosen the **Close** command from the **Control** menu on the designer or clicked the Close button.
vbFormCode	1	The **Unload** statement is invoked from code.
vbAppWindows	2	The current Windows operating environment session is ending.
vbAppTaskManager	3	The Windows **Task Manager** is closing the application.
vbFormMDIForm	4	An MDI child form is closing because the MDI form is closing.

Remarks

This event is typically used to make sure there are no unfinished tasks in the designer included in an application before that application closes. For example, if a user hasn't saved new data in any designer, the application can prompt the user to save the data.

When an application closes, you can use the QueryClose event procedure to set the **Cancel** property to **True**, stopping the closing process.

Visual Basic: RDO Data Control

Visual Studio 6.0

QueryComplete Event

See Also Example Applies To

Occurs after the query of an **rdoResultset** returns the first result set

Syntax

Private Sub *object*.**QueryComplete**(*Query* as **rdoQuery**, *ErrorOccured* as **Boolean**)

The QueryComplete event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
Query	An object expression that evaluates to an **rdoQuery** object whose query has just completed.
ErrorOccured	A Boolean expression indicating if an error occurred while processing the query.

The settings for ***ErrorOccurred*** are:

Setting	Description
True	An error occurred during query processing.
False	An error did not occur during query processing.

Fired when a query has completed. You can use this event as a notification that the result set is now ready for processing.

The ***ErrorOccured*** parameter indicates if there was an error while the query was executing. If this flag is **True**, you should check the **rdoErrors** collection for more information.

The QueryComplete event fires for all queries execute on this **rdoConnection**. This includes those queries executed via the **OpenResultset** or **Execute** methods, as well as those executed from an associated **rdoQuery** object. The ***Query*** argument is an object reference indicating which query just finished executing. Using this argument, you can write a single event handler for all queries on the connection, but still customize the handler for specific queries. When executing queries against the **rdoConnection** object itself, RDO creates an **rdoQuery** object internally, and a reference to this internal **rdoQuery** is passed as the ***Query*** argument.

This event should be used instead of polling the **StillExecuting** property to test for completion of **OpenResultset** or **Execute** method queries.

Visual Basic: RDO Data Control

RDO Events Example

This example illustrates several of the Remote Data Object (RDO) event handlers. The code establishes event variables and handlers to trap connection and query events. To help illustrate use of the BeforeConnect event, the code concatenates a workstation ID value and the current time to the end of the connect string. This permits identification of the specific connection at the server. After establishing the connection, the code executes a query that takes an fairly long time to execute the query is designed to run for about a minute. Because a 5 second QueryTimeout value is set, the QueryTimeout event should fire unless the query returns before 5 seconds has elapsed. Notice that the query itself is run asynchronously and the code does not poll for completion of the query. In this case the code simply waits for the QueryComplete or QueryTimeout events to fire indicating that the query is finished. The code also permits you to request another 5 seconds of waiting time.

Note that to make this example work correctly, you must do a number of things first, including setting references to the Remote Data Objects and Common Dialog libraries, adding a **CommandButton** and a **Timer** control to a form, plus you must change the ODBC connect string in the Form_Load() event to point to a valid server.

```
Option Explicit
Private WithEvents cn As rdoConnection
Private WithEvents EngEv As rdoEngine
Dim er As rdoError
Dim strConnect As String
Dim rs As rdoResultset
Dim TimeStart As Single
Dim clock As Integer
Dim ShowClock As Integer
Dim QueryComplete As Integer
Dim InfoMsg As String
Dim Connected As Boolean
Dim ans As Integer

Private Sub EngEv_InfoMessage()
    InfoMsg = "For your information..." _
    & " the following message" _
        & " was returned by the server." & vbCrLf
    For Each er In rdoErrors
        InfoMsg = InfoMsg & er.Number _
    & " - " & er.Description & vbCrLf
    Next

End Sub
Private Sub cn_BeforeConnect( _
    ConnectString As String, Prompt As Variant)
  InfoMsg = "About to connect to:" & ConnectString _
    & " - " & Prompt
  ConnectString = ConnectString & ";WSID=" _
    & "EventTest" & Time$ & ";"
End Sub

Private Sub cn_Connect(ByVal ErrorOccurred As Boolean)
  'Fires once connected.
  Connected = True
End Sub
```

```
Private Sub cn_Disconnect() 'Fires when disconnected
  Connected = False
End Sub

Private Sub cn_QueryComplete(ByVal Query As _
  RDO.rdoQuery, ByVal ErrorOccurred As Boolean)
  Timer1.Enabled = False
  QueryComplete = vbChecked
  RunButton.Enabled = True
  Beep

  MsgBox "Query Done"
End Sub

Private Sub cn_QueryTimeout(ByVal Query As _
  RDO.rdoQuery, Cancel As Boolean)
    ans = MsgBox("The query did not complete " _
    & "in the time allocated. " _
    & "Press Cancel to abandon the query " _
    & "or Retry to keep working.", _
        vbRetryCancel + vbQuestion, "Query Timed Out")
    If ans = vbRetry Then
        Cancel = False
        QueryComplete = vbGrayed
    Else
        Timer1.Enabled = False
        QueryComplete = vbChecked
    End If
End Sub

Private Sub MenufileExit_Click()
    cn.Close
    Unload Form1
End Sub

Private Sub RunButton_Click()
    RunButton.Enabled = False
    On Error GoTo C1EH
    QueryComplete = vbGrayed
    Timer1.Enabled = True
    Set rs = cn.OpenResultset( _
        "execute VeryLongProcedure", _
        rdOpenKeyset, rdConcurValues, rdAsyncEnable)
    TimeStart = Timer
QuitRun:
Exit Sub
C1EH:
    Debug.Print Err, Error
        InfoMsg = "Error:.. the following error" _
        & " was returned by the server." & vbCrLf
    For Each er In rdoErrors
        InfoMsg = InfoMsg & er.Number _
        & " - " & er.Description & vbCrLf
    Next
    MsgBox "Query Failed to run"
    Timer1.Enabled = False
    Resume QuitRun

End Sub
```

```
Private Sub Form_Load()
On Error GoTo FLeh
Set EngEv = rdoEngine
Set cn = New rdoConnection
Show
    With cn
        .Connect = "UID=;PWD=;database=Workdb;" _
            & "Server=SEQUEL;" _
                & "driver={SQL Server};DSN='';"
        .QueryTimeout = 5
        .CursorDriver = rdUseClientBatch
        .EstablishConnection rdDriverNoPrompt
    End With
Exit Sub

FLeh:
    Debug.Print Err, Error
    For Each er In rdoErrors
        Debug.Print er.Description
    Next
    Stop
    Resume
End Sub

Private Sub Timer1_Timer()
    Static ot As Integer
    ' Display number of seconds
    ShowClock = Int(Timer - TimeStart)
    If ShowClock = ot Then Form1.Refresh
End Sub
```

Visual Basic: RDO Data Control

Visual Studio 6.0

QueryCompleted Event (Remote Data)

See Also Example Applies To

Occurs after the query of an **rdoResultset** generated by a **RemoteData** Control returns the first result set.

Syntax

Private Sub *object*.**QueryCompleted (**[*index* **As Integer**]**)**

The QueryCompleted event syntax has these parts:

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

Remarks

When a RemoteData control completes the creation of an **rdoResultset**, the QueryCompleted event is fired. This event is not triggered if you execute the **Cancel** method which terminates processing of the query before the query has been completed.

This event fires for both asynchronous and synchronous query operations.

Visual Basic: RDO Data Control

RDO Events Example

This example illustrates several of the Remote Data Object (RDO) event handlers. The code establishes event variables and handlers to trap connection and query events. To help illustrate use of the BeforeConnect event, the code concatenates a workstation ID value and the current time to the end of the connect string. This permits identification of the specific connection at the server. After establishing the connection, the code executes a query that takes an fairly long time to execute the query is designed to run for about a minute. Because a 5 second QueryTimeout value is set, the QueryTimeout event should fire unless the query returns before 5 seconds has elapsed. Notice that the query itself is run asynchronously and the code does not poll for completion of the query. In this case the code simply waits for the QueryComplete or QueryTimeout events to fire indicating that the query is finished. The code also permits you to request another 5 seconds of waiting time.

Note that to make this example work correctly, you must do a number of things first, including setting references to the Remote Data Objects and Common Dialog libraries, adding a **CommandButton** and a **Timer** control to a form, plus you must change the ODBC connect string in the Form_Load() event to point to a valid server.

```
Option Explicit
Private WithEvents cn As rdoConnection
Private WithEvents EngEv As rdoEngine
Dim er As rdoError
Dim strConnect As String
Dim rs As rdoResultset
Dim TimeStart As Single
Dim clock As Integer
Dim ShowClock As Integer
Dim QueryComplete As Integer
Dim InfoMsg As String
Dim Connected As Boolean
Dim ans As Integer

Private Sub EngEv_InfoMessage()
    InfoMsg = "For your information..." _
    & " the following message" _
        & " was returned by the server." & vbCrLf
    For Each er In rdoErrors
        InfoMsg = InfoMsg & er.Number _
    & " - " & er.Description & vbCrLf
    Next

End Sub
Private Sub cn_BeforeConnect( _
    ConnectString As String, Prompt As Variant)
  InfoMsg = "About to connect to:" & ConnectString _
    & " - " & Prompt
  ConnectString = ConnectString & ";WSID=" _
    & "EventTest" & Time$ & ";"
End Sub

Private Sub cn_Connect(ByVal ErrorOccurred As Boolean)
  'Fires once connected.
  Connected = True
End Sub
```

```
Private Sub cn_Disconnect() 'Fires when disconnected
  Connected = False
End Sub

Private Sub cn_QueryComplete(ByVal Query As _
  RDO.rdoQuery, ByVal ErrorOccurred As Boolean)
  Timer1.Enabled = False
  QueryComplete = vbChecked
  RunButton.Enabled = True
  Beep

  MsgBox "Query Done"
End Sub

Private Sub cn_QueryTimeout(ByVal Query As _
  RDO.rdoQuery, Cancel As Boolean)
    ans = MsgBox("The query did not complete " _
    & "in the time allocated. " _
    & "Press Cancel to abandon the query " _
    & "or Retry to keep working.", _
        vbRetryCancel + vbQuestion, "Query Timed Out")
    If ans = vbRetry Then
        Cancel = False
        QueryComplete = vbGrayed
    Else
        Timer1.Enabled = False
        QueryComplete = vbChecked
    End If
End Sub

Private Sub MenufileExit_Click()
    cn.Close
    Unload Form1
End Sub

Private Sub RunButton_Click()
    RunButton.Enabled = False
    On Error GoTo C1EH
    QueryComplete = vbGrayed
    Timer1.Enabled = True
    Set rs = cn.OpenResultset( _
        "execute VeryLongProcedure", _
        rdOpenKeyset, rdConcurValues, rdAsyncEnable)
    TimeStart = Timer
QuitRun:
Exit Sub
C1EH:
    Debug.Print Err, Error
        InfoMsg = "Error:.. the following error" _
        & " was returned by the server." & vbCrLf
    For Each er In rdoErrors
        InfoMsg = InfoMsg & er.Number _
        & " - " & er.Description & vbCrLf
    Next
    MsgBox "Query Failed to run"
    Timer1.Enabled = False
    Resume QuitRun

End Sub
```

```
Private Sub Form_Load()
On Error GoTo FLeh
Set EngEv = rdoEngine
Set cn = New rdoConnection
Show
    With cn
        .Connect = "UID=;PWD=;database=Workdb;" _
            & "Server=SEQUEL;" _
                & "driver={SQL Server};DSN='';"
        .QueryTimeout = 5
        .CursorDriver = rdUseClientBatch
        .EstablishConnection rdDriverNoPrompt
    End With
Exit Sub

FLeh:
    Debug.Print Err, Error
    For Each er In rdoErrors
        Debug.Print er.Description
    Next
    Stop
    Resume
End Sub

Private Sub Timer1_Timer()
    Static ot As Integer
    ' Display number of seconds
    ShowClock = Int(Timer - TimeStart)
    If ShowClock = ot Then Form1.Refresh
End Sub
```

Visual Basic: SysInfo Control

Visual Studio 6.0

QueryChangeConfig Event

See Also Example Applies To

Occurs on a request to change the current hardware profile, either through the operating system user interface or by requesting suspend mode prior to docking or undocking the system.

Syntax

Private Sub *object*_**QueryChangeConfig(**[*index* **As Integer,**] *cancel* **As Boolean)**

The QueryChangeConfig event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	A numeric expression that evaluates to the index of a control if it is in a control array.
cancel	A Boolean value, as described in Settings.

Settings

The settings for *cancel* are:

Setting	Description
True	Prevents the system from changing the hardware profile.
False	(Default) Allows the system to change the hardware profile.

Visual Basic: RDO Data Control

Visual Studio 6.0

QueryTimeout Event

See Also Example Applies To

Occurs when the query execution time has exceeded the value set in the **QueryTimeout** property.

Syntax

Private Sub *object*.**QueryTimeout**(*Query* as **rdoQuery**, *Cancel* as **Boolean**)

The QueryTimeout event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
Query	An object expression that evaluates to an **rdoQuery** object whose query has just completed.
Cancel	A Boolean expression indicating if an error occurred while processing the query.

The settings for ***Cancel*** are:

Setting	Description
True	RDO should cancel further query processing.
False	RDO should continue processing the query for another query timeout period.

Remarks

Fired when a running query has exceeded the time specified by the **QueryTimeout** property. This event is fired each time the **QueryTimeout** time has been reached. This event is fired on both asynchronous and synchronous queries.

The ***Cancel*** parameter indicates if RDO should cancel the query or continue processing the query and wait for the number of seconds specified in the **QueryTimeout** property. The default value of this parameter is **True**, so if your code not respond to this event, the query is canceled after the **QueryTimeout** time has been reached. If the value of the parameter is set to **False**, RDO continues to wait for the query to complete for another **QueryTimeout** period.

You can use this method to display a message box to the user asking them if they wanted to cancel the query, or continue to wait another *N* seconds.

The QueryTimeout event fires for all queries execute on this **rdoConnection**. This includes those queries executed via the **OpenResultset** or **Execute** methods, as well as those executed from an associated **rdoQuery** object. The ***Query*** argument is an object reference indicating which query just timed out. Using this argument, you can write a single event handler for all queries on the connection, but still customize the handler for specific queries. When executing queries against the **rdoConnection** object itself, RDO creates an **rdoQuery** object internally, and a reference to this internal **rdoQuery** is passed as the ***Query*** argument.

Visual Basic: RDO Data Control

QueryTimeout Property, QueryTimeout Event Example

The following example sets up the query event handlers to deal with query timeout contingencies. Notice that the QueryTimeout event procedure displays a message box that permits the user to decide if they want to wait for an additional timeout period for the query. The ShowRows procedure simply dumps the rows returned.

```
Option Explicit
Dim en As rdoEnvironment
Dim cn As New rdoConnection
Dim rs As rdoResultset
Dim SQL As String
Dim col As rdoColumn
Dim er As rdoError
Public WithEvents Qd As rdoQuery

Private Sub cn_QueryTimeout( _
   ByVal Query As RDO.rdoQuery, Cancel As Boolean)
Dim ans As Integer
ans = MsgBox("Query Timed out... Press Retry to continue waiting", _
    vbRetryCancel + vbCritical, "Query Took Too Long")
If ans = vbRetry Then
    Cancel = False
Else
    Cancel = True
End If
End Sub

Private Sub RunQuery_Click()

On Error GoTo RunQueryEH

    Qd(0) = Param1
    Qd.QueryTimeout = 5
    Set rs = Qd.OpenResultset(rdOpenKeyset, _
       rdConcurReadOnly)

    If rs Is Nothing Then Else ShowRows

Exit Sub

RunQueryEH:
Debug.Print Err, Error$
    For Each er In rdoErrors
        Debug.Print er.Description, er.Number
    Next
    rdoErrors.Clear
    Resume Next

End Sub
```

```
Private Sub Form_Load()
Set en = rdoEngine.rdoEnvironments(0)
With cn
    .Connect = "uid=;pwd=;database=workdb;dsn=WorkDB;"
    .CursorDriver = rdUseClientBatch
    .EstablishConnection Prompt:=rdDriverNoPrompt
End With

Set Qd = cn.CreateQuery("LongQuery", "")
With Qd
    .SQL = "{call VeryLongStoredProcedure (?,?)}"
    .rdoParameters(1).Direction = rdParamOutput
    .rdoParameters(0).Type = rdTypeVARCHAR
End With

End Sub
```

Visual Basic Reference

Visual Studio 6.0

QueryUnload Event

See Also Example Applies To

Occurs before a form or application closes. When an **MDIForm** object closes, the QueryUnload event occurs first for the MDI form and then in all MDI child forms. If no form cancels the QueryUnload event, the Unload event occurs first in all other forms and then in an MDI form. When a child form or a **Form** object closes, the QueryUnload event in that form occurs before the form's Unload event.

Syntax

```
Private Sub Form_QueryUnload(cancel As Integer, unloadmode As Integer)

Private Sub MDIForm_QueryUnload(cancel As Integer, unloadmode As Integer)
```

The QueryUnload event syntax has these parts:

Part	Description
cancel	An integer. Setting this argument to any value other than 0 stops the QueryUnload event in all loaded forms and stops the form and application from closing.
unloadmode	A value or constant indicating the cause of the QueryUnload event, as described in Return Values.

Return Values

The *unloadmode* argument returns the following values:

Constant	Value	Description
vbFormControlMenu	0	The user chose the Close command from the Control menu on the form.
vbFormCode	1	The **Unload** statement is invoked from code.
vbAppWindows	2	The current Microsoft Windows operating environment session is ending.
vbAppTaskManager	3	The Microsoft Windows Task Manager is closing the application.
vbFormMDIForm	4	An MDI child form is closing because the MDI form is closing.
vbFormOwner	5	A form is closing because its owner is closing.

These constants are listed in the Visual Basic (VB) object library in the Object Browser.

Remarks

This event is typically used to make sure there are no unfinished tasks in the forms included in an application before that application closes. For example, if a user has not yet saved some new data in any form, your application can prompt the user to save the data.

When an application closes, you can use either the QueryUnload or Unload event procedure to set the **Cancel** property to **True**, stopping the closing process. However, the QueryUnload event occurs in all forms before any are unloaded, and the Unload event occurs as each form is unloaded.

Visual Basic Reference

QueryUnload Event Example

This example uses an **MDIForm** object containing two MDI child forms. When you choose the Close command from the Control menu to close a form, a different message is displayed than if you choose the Exit command from the File menu. To try this example, create an **MDIForm**, and then use the Menu Editor to create a File menu containing an Exit command named FileExit. Make sure that this menu item is enabled. On Form1, set the **MDIChild** property to **True**. Paste the code into the Declarations sections of the respective forms, and then press F5 to run the program.

```
' Paste into Declarations section of MDIForm1.
Private Sub MDIForm_Load ()
   Dim NewForm As New Form1   ' New instance of Form1.
   NewForm.Caption = "Form2"   ' Set caption and show.
End Sub

Private Sub FileExit_Click ()
   Unload MDIForm1   ' Exit the application.
End Sub

Private Sub MDIForm_QueryUnload (Cancel As Integer, UnloadMode As Integer)
   Dim Msg   ' Declare variable.
   ' Set the message text.
   Msg = "Do you really want to exit the application?"
   ' If user clicks the No button, stop QueryUnload.
   If MsgBox(Msg, vbQuestion + vbYesNo, Me.Caption) = vbNo Then Cancel = True
End Sub

' Paste into Declarations section of Form1.
Private Sub Form_QueryUnload (Cancel As Integer, UnloadMode As Integer)
   Dim Msg   ' Declare variable.
   If UnloadMode > 0 Then
      ' If exiting the application.
      Msg = "Do you really want to exit the application?"
   Else
      ' If just closing the form.
      Msg = "Do you really want to close the form?"
   End If
   ' If user clicks the No button, stop QueryUnload.
   If MsgBox(Msg, vbQuestion + vbYesNo, Me.Caption) = vbNo Then Cancel = True
End Sub
```