Visual Basic Reference

Visual Studio 6.0

Unformat Event

See Also Example Applies To

Occurs after the **StdFormat** object unformats the value.

Syntax

Sub *object*_**Unformat**(**ByRef** d*atavalue* **As StdDataValue**)

The Unformat event syntax has these parts:

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

Remarks

This event allows the user to do unformatting that the standard settings of the **StdDataFormat** object cannot accomplish.

Visual Basic: Page Designer

Visual Studio 6.0

Unload Event (DHTMLPage)

See Also Example Applies To

Occurs in response to the user navigating away from a given HTML page or closing the browser.

Syntax

Private Sub *object*_**Unload**

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

Remarks

You can use the **Unload** event to clean up code before execution is terminated. When the application fires this event, you may wish to clean up by closing open files, deleting temporary files, and so on. All of the objects on the HTML page still exist when the **Unload** event fires.

Visual Basic Reference

Visual Studio 6.0

Unload Event

See Also Example Applies To

Occurs when a form is about to be removed from the screen. When that form is reloaded, the contents of all its controls are reinitialized. This event is triggered by a user closing the form using the Close command on the Control menu or an **Unload** statement.

Syntax

```
Private Sub object_Unload(cancel As Integer)
```

The Unload event syntax has these parts:

Part	Description
Object	An object expression that evaluates to an object in the Applies To list.
Cancel	Integer that determines whether the form is removed from the screen. If *cancel* is 0, the form is removed. Setting *cancel* to any nonzero value prevents the form from being removed.

Remarks

Setting *cancel* to any nonzero value prevents the form from being removed, but doesn't stop other events, such as exiting from the Microsoft Windows operating environment. Use the QueryUnload event to stop exiting from Windows.

Use an Unload event procedure to verify that the form should be unloaded or to specify actions that you want to take place when the form is unloaded. You can also include any form-level validation code you may need for closing the form or saving the data in it to a file.

The QueryUnload event occurs before the Unload event. The Unload event occurs before the Terminate event.

The Unload event can be caused by using the **Unload** statement, or by the user choosing the Close command on a form's Control menu, exiting the application with the End Task button on the Windows Task List, closing the MDI form for which the current form is a child form, or exiting the Microsoft Windows operating environment while the application is running.

Visual Basic Reference

Unload Event Example

This example demonstrates a simple procedure to close a form while prompting the user with various message boxes. In an actual application, you can add calls to general purpose **Sub** procedures that emulate the processing of the Exit, Save, and Save As commands on the File menu in Visual Basic. To try this example, paste the code into the Declarations section of a form, and then press F5. Once the form is displayed, press ALT+F4 to close the form.

```
Private Sub Form_Unload (Cancel As Integer)
   Dim Msg, Response   ' Declare variables.
   Msg = "Save Data before closing?"
   Response = MsgBox(Msg, vbQuestion + vbYesNoCancel, "Save Dialog")
   Select Case Response
      Case vbCancel   ' Don't allow close.
         Cancel = -1
         Msg = "Command has been canceled."
      Case vbYes
      ' Enter code to save data here.
         Msg = "Data saved."'
      Case vbNo
         Msg = "Data not saved."
   End Select
   MsgBox Msg, vbOKOnly, "Confirm"   ' Display message.
End Sub
```

Visual Basic: Windows Controls

Visual Studio 6.0

UpClick Event

See Also Example Applies To

This event occurs when the up or right arrow button is clicked.

Private Sub *object*_**UpClick(**[*index* as integer]**)**

The UpClick 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

The UpClick event occurs after the Change event.

Visual Basic Reference

Visual Studio 6.0

Updated Event

See Also Example Applies To

Occurs when an object's data has been modified.

Syntax

Sub *object*_**Updated (***code* **As Integer)**

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
code	An integer that specifies how the object was updated, as described in Settings.

Settings

The settings for *code* are:

Constant	Value	Description
vbOLEChanged	0	The object's data has changed.
vbOLESaved	1	The object's data has been saved by the application that created the object.
vbOLEClosed	2	The file containing the linked object's data has been closed by the application that created the object.
vbOLERenamed	3	The file containing the linked object's data has been renamed by the application that created the object.

Remarks

These constants are listed in the Visual Basic (VB) object library in the Object Browser. You can use this event to determine if an object's data has been changed since it was last saved. To do this, set a global variable in the Updated event indicating that the object needs to be saved. After you save the object, reset the variable.

Visual Basic Reference

Visual Studio 6.0

UserEvent Event

See Also Example Applies To

Occurs in response to the firing of a run-time defined event.

Syntax

Private Sub *object*_**UserEvent**(*eventname* **As String**)

The UserEvent event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
eventname	Contains a custom event name that was added at run time.

Remark

When receiving an event name that was not defined at design time, the **URLFor** method returns a URL which when processed by the client fires the UserEvent event on the respective **WebItem**. The UserEvent event will be passed the name of the run-time defined event. Using the **URLFor** method you can generate a URL that, if retrieved by the browser in response to a user action, will invoke the **UserEvent** method. The **WebClass** should take appropriate action according to the meaning of the event. You write the event procedures for all such events in your **WebClass** within the UserEvent event. If you have a **WebClass** that contains several user events, use a conditional statement such as **If** or **Select Case** to specify the actions the system should perform for each.

Visual Basic Reference

Visual Studio 6.0

Validate Event (Data Control)

See Also Example Applies To

Occurs before a different record becomes the current record; before the **Update** method (except when data is saved with the **UpdateRecord** method); and before a **Delete**, **Unload**, or **Close** operation.

Syntax

Private Sub *object*_**Validate (**[*index* **As Integer**,] *action* **As Integer**, *save* **As Integer)**

The Validate 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.
action	An integer that indicates the operation causing this event to occur, as described in Settings.
save	A Boolean expression specifying whether bound data has changed, as described in Settings.

Settings

The settings for *action* are:

Constant	Value	Description
vbDataActionCancel	0	Cancel the operation when the **Sub** exits
vbDataActionMoveFirst	1	**MoveFirst** method
vbDataActionMovePrevious	2	**MovePrevious** method
vbDataActionMoveNext	3	**MoveNext** method
vbDataActionMoveLast	4	**MoveLast** method
vbDataActionAddNew	5	**AddNew** method

vbDataActionUpdate	6	**Update** operation (not **UpdateRecord**)
vbDataActionDelete	7	**Delete** method
vbDataActionFind	8	**Find** method
vbDataActionBookmark	9	The **Bookmark** property has been set
vbDataActionClose	10	The **Close** method
vbDataActionUnload	11	The form is being unloaded

The settings for *save* are:

Setting	Description
True	Bound data has changed
False	Bound data has not changed

Remarks

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

The *save* argument initially indicates whether bound data has changed. This argument can still be **False** if data in the copy buffer is changed. If *save* is **True** when this event exits, the **Edit** and **UpdateRecord** methods are invoked. Only data from bound controls or from the copy buffer where the **DataChanged** property is set to **True** are saved by the **UpdateRecord** method.

This event occurs even if no changes have been made to data in bound controls and even if no bound controls exist. You can use this event to change values and update data. You can also choose to save data or stop whatever action is causing the event to occur and substitute a different action.

You can change the *action* argument to convert one action into another. You can change the various Move methods and the **AddNew** method, which can be freely exchanged (any Move into **AddNew**, any Move into any other Move, or **AddNew** into any Move). When using **AddNew**, you can use **MoveNext** and then execute another **AddNew** to examine the **EditMode** property to determine if an **Edit** or **AddNew** operation is in progress. Attempting to change **AddNew** or one of the Moves into any of the other actions is either ignored or produces a trappable error. Any action can be stopped by setting *action* to 0.

In your code for this event, you can check the data in each bound control where **DataChanged** is **True**. You can then set **DataChanged** to **False** to avoid saving that data in the database.

You can't use any methods (such as **MoveNext**) on the underlying **Recordset** object during this event.

Visual Basic Reference

DataChanged Property and Validate Event Example

This example illustrates simple data validation. In the Authors table in the Biblio.mdb database, there are two fields: Au_ID and Author. Since the value in Au_ID is used to uniquely identify the author, this value should not change. The example doesn't allow changes to the Au_ID field, which is bound to `Text1`.

```
Private Sub Data1_Validate (Action As Integer, Save As Integer)
   If Text1.DataChanged Then   ' Check for change in data.
      MsgBox "You can't change the ID number."
      Text1.DataChanged = False   ' Don't save changed data.
   End If
   ...
End Sub
```

Visual Basic: RDO Data Control

Visual Studio 6.0

Validate Event (Remote Data)

See Also Example Applies To

Occurs before a different row becomes the current row; before the **Update** method (except when data is saved with the **UpdateRow** method); and before a **Delete**, **Unload**, or **Close** operation.

Syntax

Private Sub *object*_**Validate (**[*index* **As Integer**,] *action* **As Integer**, *reserved* **As Integer)**

The Validate 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.
Action	An Integer or constant that indicates the operation causing this event to occur, as described in Settings.
Reserved	An Integer argument preserved only for compatibility with RDO version 1.0 and ignored by the Remote Data control.

Settings

The settings for *action* are:

Constant	Value	Description
RdActionCancel	0	Cancel the operation when the **Sub** exits.
RdActionMoveFirst	1	**MoveFirst** method.
RdActionMovePrevious	2	**MovePrevious** method.
RdActionMoveNext	3	**MoveNext** method.
RdActionMoveLast	4	**MoveLast** method.
RdActionAddNew	5	**AddNew** method.

RdActionUpdate	6	**Update** operation (not **UpdateRow**).
RdActionDelete	7	**Delete** method.
RdActionFind	8	**Find** method (not implemented).
RdActionBookmark	9	The **Bookmark** property has been set.
RdActionClose	10	The **Close** method.
RdActionUnload	11	The form is being unloaded.
RdActionUpdateAddNew	12	A new row was inserted into the result set.
RdActionUpdateModified	13	The current row changed.
RdActionRefresh	14	**Refresh** method executed.
RdActionCancelUpdate	15	Update canceled.
RdActionBeginTransact	16	**BeginTrans** method.
RdActionCommitTransact	17	**CommitTrans** Method.
RdActionRollbackTransact	18	**RollbackTrans** Method
RdActionNewParameters	19	Change in parameters, or order of columns or rows.
RdActionNewSQL	20	SQL statement changed.

Remarks

This event can occur regardless of whether data in bound controls changes, or whether bound controls exist. You can use this event to change values and update data. You can also choose to save data or stop whatever action is causing the event to occur and substitute a different action.

If you attempt to change **AddNew** or one of the *Move* methods into any of the other actions (any *Move* into **AddNew**, any *Move* into any other *Move*, or **AddNew** into any *Move*), it results in the action being halted, the same as if you set it to *rdActionCancel*.

In your code for this event, you can check the data in each bound control where **DataChanged** is **True**. You can then set **DataChanged** to **False** to avoid saving that data in the database.

Note Because a data-aware control can have more than one bound property, the **DataChanged** property must be examined for each of the bound properties as enumerated in the **Bindings** collection.

You can't use any methods (such as **MoveNext**) on the underlying **rdoResultset** object during this event.

Visual Basic Reference

Visual Studio 6.0

Validate Event

See Also Example Applies To

Occurs before the focus shifts to a (second) control that has its **CausesValidation** property set to **True**.

Syntax

```
Private Sub object_Validate(KeepFocus As Boolean)
```

The Validate event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
KeepFocus	Value that determines if the control loses focus. Setting *KeepFocus* to **True** specifies that the control retains the focus.

Remarks

The Validate event works in tandem with the **CausesValidation** property to prevent a control from losing the focus until certain criteria are met.

Important The Validate event only occurs when the control which is about to receive the focus has its **CausesValidation** property set to **True**.

Visual Basic Reference

Validate Event, CausesValidation Property Example

The example uses three controls to demonstrate the use of the Validate event and **CausesValidation** property. By default, the **CausesValidation** property of the two TextBox controls are set to **True**. Thus when you try to shift the focus from one TextBox to the other, the Validate event occurs. If Text1 doesn't contain a date, or if Text2 doesn't contain a number larger than 10, the shift of focus is prevented. Because the **CausesValidation** property of the Command1 control is set to **False**, however, you can always click the Help button.

To try the example, place one **CommandButton** and two **TextBox** controls on a form. Paste the code into the Declarations section of the form and run the project. Attempt to shift the focus by pressing the Tab key.

```
Private Sub Form_Load()
   ' Set the button's CausesValidation property to False. When the user
   ' clicks the button, the Validate event does not occur.
   ' Set the Caption property of the button to "Help".
   With Command1
      .CausesValidation = False
      .Caption = "Help"
   End With

   Show
   With Text1 ' Select text of Text1 and set focus to it.
      .SelLength = Len(Text1.Text)
      .SetFocus
   End With
End Sub

Private Sub Command1_Click()
   ' Give the user help when the button is clicked.
   MsgBox _
   "Text1 must be set to a date." & VbCrLF & _
   "Text2 must be a number less than 10."
End Sub

Private Sub Text1_Validate(KeepFocus As Boolean)
   ' If the value is not a date, keep the focus, unless the user
   ' clicks Help.
   If Not IsDate(Text1.Text) Then
      KeepFocus = True
      MsgBox "Please insert a date in this field.", ,   "Text1"
   End if
End Sub

Private Sub Text2_Validate(KeepFocus As Boolean)
   ' If the value is a number larger than 10, keep the focus.
   If Not IsNumeric(Text2.Text) Or Val(Text2.Text) > 10 Then
      KeepFocus = True
MsgBox _
"Please insert a number less than or equal to 10.", , "Text2"
```

```
    End If
End Sub
```

Visual Basic: MaskedEdit Control

Visual Studio 6.0

ValidationError Event

See Also Example Applies To

Occurs when the **Masked Edit** control receives invalid input, as determined by the input mask.

Syntax

Private Sub *ctlname*_**ValidationError**(*InvalidText* **As String**; *StartPosition* **As Integer**)

Remarks

InvalidText is the value of the **Text** property, including the invalid character. This means that any placeholders and literal characters used in the input mask are included in *InvalidText*.

StartPosition is the position in *InvalidText* where the error occurred (the first invalid character).

Visual Basic: DataRepeater Control

Visual Studio 6.0

VisibleRecordsChanged Event

See Also Example Applies To

Occurs when the **VisibleRecords** property changes.

Syntax

*object*_**VisibleRecordsChanged()**

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

Visual Basic: RDO Data Control

Visual Studio 6.0

WillAssociate Event

See Also Example Applies To

Occurs before a new connection is associated with the object.

Private Sub *object*.**WillAssociate**(*Connection* as **rdoConnection,** *Cancel* as **Boolean**)

The WillAssociate event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
Connection	An object expression that evaluates to the **rdoConnection** object that is to be associated.
Cancel	A Boolean expression indicating if RDO should prohibit the association.

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

Setting	Description
True	RDO will prohibit the association.
False	(Default) RDO will not prohibit the association.

Remarks

This event is raised after you set the **ActiveConnection** property to a valid **rdoConnection** object, but before the actual associate is made.

The ***Connection*** argument is a reference to the **rdoConnection** object that you are attempting to associate with the **rdoResultset** object. When the WillAssociate event is raised, the **ActiveConnection** property remains set to the value *before* the attempted association. You can use this property to determine the current **rdoResultset** connection association.

You can prohibit the association by setting the ***Cancel*** argument to **True**, causing RDO to not associate the result set with the new connection and produce a runtime error. If you do not prohibit the association using the ***Cancel*** argument, the **ActiveConnection** property is set to the reference contained in the ***Connection*** parameter after this event procedure completes.

Visual Basic: RDO Data Control

Visual Studio 6.0

WillDissociate Event

See Also Example Applies To

Occurs before the connection is set to nothing.

Private Sub *object*.**WillDissociate**(*Cancel* as **Boolean**)

The WillDissociate event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
Cancel	A Boolean expression indicating whether RDO should prohibit the disassociation.

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

Setting	Description
True	RDO will prohibit the change.
False	(Default) RDO will not prohibit the change.

Remarks

This event is raised when the developer attempts to set the **ActiveConnection** property to **Nothing** but *before* the result set is dissociated from the connection.

If you wish to prohibit the dissociation, set the ***Cancel*** parameter to **True**, causing RDO to cancel the operation and trigger a trappable error.

The default value for the ***Cancel*** parameter is **False**, so if the event is not trapped, the dissociation is completed.

Visual Basic: RDO Data Control

Visual Studio 6.0

WillExecute Event

See Also Example Applies To

Occurs before the execution of a query,

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

The WillExecute 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 RDO should prohibit the change.

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

Setting	Description
True	RDO will prohibit the change.
False	(Default) RDO will not prohibit the change.

Remarks

This event is fired before the execution of a query, regardless if it is an action or row-returning query. You can trap this event to disallow the execution of certain queries, or to make last-minute adjustments to the **rdoQuery** object's SQL string.

The ***Cancel*** argument allows you to disallow the query. The ***Cancel*** parameter will default to **False**, but if you set it to **True**, the query will not execute, and RDO generates a trappable error indicating that the query was canceled.

For example, you can pre-screen the query to make sure the WHERE clause will not cause a table-scan operation. Thus, by setting the ***Cancel*** argument to **True**, you can prohibit users from searching for customers with the last name of Smith without also providing a first name or street address.

The WillExecute 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 is about to execute. 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

Visual Studio 6.0

WillChangeData Event

See Also Example Applies To

Occurs before data is changed in the column.

Private Sub *object*.**WillChangeData**(*NewValue* as **Variant,** *Cancel* as **Boolean**)

The WillChangeData event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
NewValue	A **Variant** expression containing the data to be applied to the column.
Cancel	A Boolean expression indicating if RDO should prohibit the change.

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

Setting	Description
True	RDO will prohibit the change.
False	(Default) RDO will not prohibit the change.

Remarks

This event is raised just before RDO commits any change to the data in a column. By trapping this event, you can either modify the new value, or prohibit the change by modifying the ***Cancel*** argument.

If you modify the ***NewValue*** parameter, the modified value is assigned to the columns **Value** property. This allows you translate or substitute data.

By default, the ***Cancel*** argument is **False**, but if you set it to **True**, the change to the columns data is canceled, and RDO generates a trappable error.

Visual Basic: RDO Data Control

Visual Studio 6.0

WillUpdateRows Event

See Also Example Applies To

Occurs before an update to the database occurs.

Private Sub *object*.**WillUpdateRows**(*ReturnCode* as **Integer**)

The WillUpdateRows event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
ReturnCode	An Integer expression or constant indicating whether the developer handled the update or not as described in Settings.

The ***ReturnCode*** argument is used to notify RDO about what your code did during the event handling. The possible values for this argument are as follows:

Setting	Description
rdUpdateSuccessful	Your code handled the update and was successful in doing so.
rdUpdateWithCollisions	Your code handled the update successfully, but some rows produced collisions (batch mode only).
rdUpdateFailed	Your code attempted to handle the update, but encountered an error when doing so.
rdUpdateNotHandled	Your code did not handle the update. RDO should continue notifying and if no one handles the update, RDO should update the data itself.

Remarks

The WillUpdateRows event is raised before updated, new and deleted rows are committed to the server. You can override the update behavior of the cursor by responding to this event and perform your own updates using stored procedures or any other mechanism you choose.

If the result set is using batch optimistic concurrency, this event is only raised when the **BatchUpdate** method is called. In this case, the entire set of changes is about to be transmitted to the server.

If the result set is not in a batch mode, the WillUpdateRows event is raised for each call to the **Update** method, since the changes for that row are immediately sent to the server.

To summarize, no matter what mode the result set is in, this event is only raised before data is actually sent to the server.

If you set the ***ReturnCode*** argument to **rdUpdateSuccessful**, RDO assumes that your code successfully handled the update. RDO will not send this event to any additional clients (if there is more than one handler of this event) and the status for the row(s) and their columns is set to **rdRowUnmodified** and **rdColUnmodified** respectively.

If you set the ***ReturnCode*** parameter to **rdUpdateWithCollisions**, RDO assumes that you have successfully handled the update, but some rows caused collisions. RDO will not send this event to any additional clients (if there was more than one handler of this event) and the status for the rows and their columns is not changed. It is your code's responsibility to set the column status flags during the handling of this event. The **rdUpdateWithCollisions** would only be used if you are using batch optimistic concurrency and you wanted to check for and handle collisions in code.

If the developer sets the ***ReturnCode*** parameter to **rdUpdateFailed**, RDO assumes that your code attempted to handle the update, but encountered an error while doing so. RDO will not send this event to any additional clients (if there was more than one handler of this event) and the status for the row(s) and their columns remains unchanged. Finally, RDO generates a runtime error to be trapped by the **Update** method causing the WillUpdate event to fire.

If you set the ***ReturnCode*** parameter to **rdUpdateNotHandled**, RDO will assume that the developer did not handle the update, and RDO will continue to raise this event to all remaining clients (if there was more than one handler of this event). If all clients return rdUpdateNotHandled, RDO will perform the update itself, according the normal rules.

The default value for the ***ReturnCode*** parameter is **rdUpdateNotHandled**, so if no client sinks the event, or no client changes the value of ***ReturnCode***, RDO will perform the update.

Visual Basic Reference

Visual Studio 6.0

WriteProperties Event

See Also Example Applies To

Occurs when an instance of an object is to be saved. This event signals to the object that the state of the object needs to be saved, so that the state can be restored later. In most cases, the state of the object consists only of property values.

Syntax

Sub *object*_**WriteProperties(***pb* **As PropertyBag)**

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
pb	An object of the type PropertyBag class to write the data to.

Remarks

The author of *object* can have *object* save the state when the WriteProperties event occurs, by calling the **WriteProperty** method of the **PropertyBag** object for each value that is to be saved.

Note The *pb* property bag may be different from the *pb* that was passed to the most recent ReadProperties event.

The WriteProperties event may occur multiple times during the life of an instance of *object*.