Visual Basic: DataGrid Control

Visual Studio 6.0

HeadClick Event

See Also Example Applies To

Occurs when the user clicks on the header for a particular column of a **DataGrid** control.

Syntax

Private Sub *object*_**HeadClick (**[*index* **As Integer**,] *colindex* **As Integer)**

The HeadClick 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.
colindex	An integer that identifies the column.

Remarks

One possible use for this event is to resort the **Recordset** object based on the selected column.

Visual Basic: DataGrid Control

HeadClick Event Example

This example sorts the record source of the **Data** control based on which column the user clicked.

```
Private Sub DataGrid1_HeadClick (ColIndex As Integer)
   Data1.RecordSource = "Select * From Publishers Order By " & _
    DataGrid1.Columns(ColIndex).DataField
   Data1.Refresh
End Sub
```

Visual Basic: Windows Controls

Visual Studio 6.0

HeightChanged Event

See Also Example Applies To

Occurs when the **Height** of a **CoolBar** control changes.

Syntax

Private Sub *object*_**HeightChanged([***index* **As Integer]**, *newheight* **As Single)**

The HeightChanged event syntax has these parts:

Part	Description
object	An object expression that evaluates to a CoolBar control.
index	An integer that uniquely identifies a control if its in a control array.
newheight	Single-precision floating-point number specifying the new height of the control.

Remarks

The **HeightChanged** event occurs after the **Resize** event when the **CoolBar** height needs to change. This can occur when the user rearranges bands at run time, or when the height of one or more bands is changed programmatically.

This event is useful when the **CoolBar** is hosted by a container which is capable of suppressing changes. In such a case, reading the **Height** property during the **Resize** event may not be reliable, or the **Resize** event may be suppressed. The **HeightChanged** event allows you to add code to allow the **CoolBar** control to display properly.

Visual Basic Reference

Visual Studio 6.0

Hide Event (UserControl Object)

See Also Example Applies To

Occurs when the objects **Visible** property changes to **False**.

Syntax

Sub *object*_**Hide()**

The Hide event syntax has these parts:

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

Remarks

In order to draw to the screen in Windows, any object must have a window, temporarily or permanently; Visual Basic ActiveX controls have permanent windows. Before a control has been sited on a form, its window is not on the container. The control receives Hide events when the window is removed.

While the controls window is on the form, the object receives a Hide event when the controls **Visible** property changes to **False**.

The control does *not* receive Hide events if the form is hidden and then shown again, or if the form is minimized and then restored. The controls window remains on the form during these operations, and its **Visible** property doesnt change.

If the control is being shown in an internet browser, a Hide event occurs when the page is moved to the history list.

Note If the control is used with earlier versions of Visual Basic than 5.0, the control will not receive Hide events at design time. This is because earlier versions of Visual Basic did not put any visible windows on a form at design time.

Visual Basic Reference

Visual Studio 6.0

Hide Event (UserDocument Object)

See Also Example Applies To

Occurs when the objects **Visible** property changes to **False**.

Syntax

Sub *object*_**Hide()**

The Hide event syntax has these parts:

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

Remarks

In order to draw to the screen in Windows, any object must have a window, temporarily or permanently. Visual Basic ActiveX documents have permanent windows. The **UserDocument** object receives Hide events when the window is removed.

While *objects* window is on the container, *object* receives a Hide event when *objects* **Visible** property changes to **False**.

Object does *not* receive Hide events if the container is hidden and then shown again, or if the container is minimized and then restored. *Objects* window remains on the container during these operations, and its **Visible** property doesnt change.

If *object* is being shown in an internet browser, a Hide event occurs when the page is moved to the history list by navigating off *object* to another document, or when Internet Explorer 3.0 is terminated while *object* is being viewed or is still within the cache of active documents. Use the event to destroy any global object references before navigating to another document.

Note If *object* is used with earlier versions of Visual Basic than 5.0, *object* will not receive Hide events at design time. This is because earlier versions of Visual Basic did not put any visible windows on a form at design time.

Visual Basic Reference

Visual Studio 6.0

HitTest Event

See Also Example Applies To

Occurs when the user moves the mouse over a UserControl object. Only occurs when the **Windowless** property of the UserControl is set to True and the **BackStyle** property is set to Transparent.

Syntax

Private Sub *object*_**HitTest(***x* **As Single,** *y* **As Single,** *HitResult* **As Integer)**

The HitTest event syntax has these parts:

Part	Description
object	A UserControl object.
x, y	A number that specifies the current location of the mouse pointer. The x and y values are always expressed in terms of the coordinate system set by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties of the object.
HitResult	An integer that specifies hit behavior, as described in Settings.

Settings

The settings for *HitResult* are:

Constant	Setting	Description
vbHitResultOutside	0	The mouse pointer is outside of the visible area of the control. Mouse messages are forwarded.
vbHitResultTransparent	1	The mouse pointer is over a transparent region of the control. Mouse messages can be forwarded. Must be set at run time.
vbHitResultClose	2	The mouse pointer is close to the visible area of the control. The close region must be defined by the developer at design time. Mouse messages can be forwarded. Must be set at run time.
vbHitResultHit	3	The mouse pointer is over the visible area of the control. The control receives mouse messages.

Remarks

The HitTest event is used to determine whether a UserControl will receive mouse events such as MouseUp, MouseDown, MouseOver, Click, and DblClick. HitTest occurs before any other mouse messages.

By default, a UserControl will return a HitResult of 0 when the mouse pointer is over a transparent area of the control, or a HitResult of 3 when over the MaskRegion. The MaskRegion is defined by a combination of the **MaskPicture** and **MaskColor** properties.

In the HitTest event you can change the hit testing behavior of your control by assigning a different value to the HitResult. For example, if you have a UserControl with a transparent area in its center, its possible that another control may be visible underneath that area. By changing the HitResult to 1 or 2 in the HitTest event, you can allow the control underneath to receive the mouse messages. If there is nothing underneath at run time or if the control underneath declines the hit, your control will get a second or third chance to receive the mouse messages.

Hit testing is performed in the following order for multiple overlapping controls:

- The topmost control in the ZOrder that returns a HitResult of 3 (vbHitResultHit) will receive mouse messages.
- If no control in the ZOrder returns a hit, the topmost control that returns a HitResult of 2 (vbHitResultClose) will receive mouse messages.
- If no control returns a hit, the topmost control that returns a HitResult of 1 (vbHitResultTransparent) will receive mouse messages.
- If no control returns a hit, the mouse messages are forwarded to the underlying container.

The HitTest event only occurs when the **Windowless** property of the UserControl has been set to True and the **BackStyle** property has been set to Transparent. If the **Windowless** property is False or the **BackStyle** property equals Opaque, any code in the HitTest event procedure will be ignored.

Note The order of hit testing described above applies to UserControl objects in a Visual Basic container. Other containers may not perform hit testing in the same order.

Visual Basic: RDO Data Control

Visual Studio 6.0

InfoMessage Event

See Also Example Applies To

Occurs when informational messages are added to the **rdoErrors** collection.

Private Sub *object*.**InfoMessage**( )

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

Remarks

This event is raised after RDO receives a SQL_SUCCESS_WITH_INFO return code from the ODBC Driver Manager, and populates the **rdoErrors** collection with the informational messages.

The InfoMessage event is raised once for each *set* of informational messages. Thus, if an RDO method generates several informational messages, this event is raised only once after the last message has been added to the collection. You can trap this event and examine the contents of the **rdoErrors** collection and decide what action is appropriate.

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 Reference

Visual Studio 6.0

Initialize Event

See Also Example Applies To

Occurs when an application creates an instance of a **Form**, **MDIForm**, **User** control, **Property Page**, **Webclass**, **DHTML Page Designer**, or class.

Syntax

Private Sub *object*_**Initialize( )**

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

Remarks

You trigger the Initialize event when you:

- Use the **CreateObject** function to create an instance of a class. For example:

  ```
  Set X = CreateObject("Project1.MyClass")
  ```

- Refer to a property or event of an automatically created instance of a form or class in your code. For example:

  ```
  MyForm.Caption = "Example"
  ```

Use this event to initialize any data used by the instance of the **Form**, **MDIForm**, or class. For a **Form** or **MDIForm**, the Initialize event occurs before the Load event.

Visual Basic Reference

Visual Studio 6.0

InitProperties Event

See Also Example Applies To

Occurs when a new instance of an object is created.

Syntax

Sub *object*_**InitProperties()**

The InitProperties event syntax has this part:

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

Remarks

This event allows the author of the object to initialize a new instance of the object. This event occurs only when a new instance of an object is being created; this is to allow the author of the object to distinguish between creating a new instance of the object and loading an old instance of the object.

By putting in code to initialize new instances in the InitProperties event rather than the Initialize event, the author can avoid cases where loading data through a ReadProperties event into an old instance of the object will undo the initialization of the object.

Visual Basic Extensibility Reference

Visual Studio 6.0

ItemActivated Event

See Also Example Applies To

Occurs when a component is double-clicked in the Project window, and when a project is single-clicked in a project window when there are multiple projects loaded in the IDE.

Syntax

Sub *object*_**ItemActivated(***vbcomponent* **As VBComponent)**

Sub *object*_**ItemActivated(***vbproject* **As VBProject)**

The ItemActivated event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
vbcomponent	A **VBComponent** object specifying the name of the component that was double-clicked.
vbproject	A **VBProject** object specifying the name of the project which was double-clicked.

Visual Basic Extensibility Reference

Visual Studio 6.0

ItemAdded Event (VBA Add-In Object Model)

See Also Example Applies To Specifics

Occurs after a reference is added.

Syntax

Sub *object*_**ItemAdded(ByVal** *item* **As Reference)**

The required *item* argument specifies the item that was added.

Remarks

The ItemAdded event occurs when a **Reference** is added to the **References** collection.

Visual Basic Extensibility Reference

Visual Studio 6.0

ItemAdded Event

See Also Example Applies To

Occurs after a project, control, or component is added to the current project.

Syntax

Sub *object*_**ItemAdded (***vbproject* **As VBProject)**

Sub *object*_**ItemAdded (***vbcomponent* **As VBComponent)**

Sub *object*_**ItemAdded (***vbcontrol* **As VBControl)**

The ItemAdded event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
vbproject	A **VBProject** object specifying the name of the project that was loaded.
vbcomponent	A **VBComponent** object specifying the name of the component that was loaded.
vbcontrol	A **VBControl** object specifying the name of the control that was loaded.

Visual Basic: Windows Controls

Visual Studio 6.0

ItemClick Event

See Also Example Applies To

Occurs when a **ListItem** object in a **ListView** control is clicked.

Syntax

```
Private Sub object_ItemClick(ByVal Item As ListItem)
```

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

Part	Description
object	An object expression that evaluates to a **ListView** control.
listitem	The **ListItem** object that was clicked.

Remarks

Use this event to determine which **ListItem** was clicked. This event is triggered before the **Click** event. The standard **Click** event is generated if the mouse is clicked on any part of the **ListView** control. The **ItemClick** event is generated only when the mouse is clicked on the text or image of a **ListItem** object.

Visual Basic: Windows Controls

ItemClick Event Example

This example populates a **ListView** control with contents of the Publishers table in the Biblio.mdb database. When a **ListItem** object is clicked, the code checks the value of the **Index** property. If the value is less than 15, nothing occurs. If the value is greater than 15, the **ListItem** object is ghosted. To try the example, place a **ListView** control on a form and paste the code into the form's Declarations section. Run the example and click on one of the items.

```
Private ListView1_ItemClick(ByVal Item As ListItem)
   Select Case Item.Index
   Case Is = <15
      Exit Sub
   Case Is => 15
      ' Toggle Ghosted property.
      Item.Ghosted = Abs(Item.Ghosted) - 1
   End Select
End Sub

Private Sub Form_Load()
   ' Create an object variable for the ColumnHeader object.
   Dim clmX As ColumnHeader
   ' Add ColumnHeaders. The width of the columns is the width
   ' of the control divided by the number of ColumnHeader objects.
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Company", ListView1.Width / 3)
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Address", ListView1.Width / 3)
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Phone", ListView1.Width / 3)

   ListView1.BorderStyle = ccFixedSingle ' Set BorderStyle property.

   ' Create object variables for the Data Access objects.
   Dim myDb As Database, myRs As Recordset
   ' Set the Database to the BIBLIO.MDB database.
   Set myDb = DBEngine.Workspaces(0).OpenDatabase("BIBLIO.MDB")
   ' Set the recordset to the Publishers table.
   Set myRs = myDb.OpenRecordset("Publishers", dbOpenDynaset)

   ' Create a variable to add ListItem objects.
   Dim itmX As ListItem

   ' While the record is not the last record, add a ListItem object.
   ' Use the Name field for the ListItem object's text.
   ' Use the Address field for the ListItem object's SubItem(1).
   ' Use the Phone field for the ListItem object's SubItem(2).

   While Not myRs.EOF

      Set itmX = ListView1.ListItems.Add(, , CStr(myRs!Name))

      ' If the Address field is not Null, set SubItem 1 to the field.
      If Not IsNull(myRs!Address) Then
         itmX.SubItems(1) = CStr(myRs!Address) ' Address field.
```

```
      End If

      ' If the Phone field is not Null, set the SubItem 2 to the field.
      If Not IsNull(myRs!Telephone) Then
         itmX.SubItems(2) = myRs!Telephone  ' Phone field.
      End If

      myRs.MoveNext ' Move to next record.
   Wend
   ListView1.View = lvwReport   ' Set View to Report.
End Sub

Private Sub ListView1_ColumnClick(ByVal ColumnHeader As ColumnHeader)
   ListView1.SortKey = ColumnHeader.Index - 1
   ListView1.Sorted = True
End Sub
```

Visual Basic Reference

Visual Studio 6.0

ItemCheck Event (ListBox Control)

See Also Example Applies To

Occurs when a **ListBox** control **Style** property is set to 1 (checkboxes) and an items checkbox in the **ListBox** control is selected or cleared.

Syntax

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

The ItemCheck 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 the item in the listbox which was clicked.

Remarks

Note The ItemCheck event does not occur when a list item is only highlighted; rather, it occurs when the check box of the list item is selected or cleared.

The ItemCheck event can also occur programmatically whenever an element in Selected array of the **ListBox** is changed (and its **Style** property is set to 1.)

The ItemCheck event occurs before the Click event.

Attempting to unload a form using **Unload Me** in a **ListBox** control's ItemCheck event causes a General Protection Fault (GPF). For example:

```
Private Sub Form_Load()
     Dim i%
     For i = 0 To 20
       List1.AddItem i
     Next i
   End Sub

Private Sub List1_ItemCheck(Item As Integer)
  Unload Me
End Sub
```

This example also produces a GPF if the spacebar is pressed, or if you attempt to select an item in the **ListBox** using a **CommandButton**.

The recommended procedure for unloading a form is to add the **Unload** statement to the Click event in a **CommandButton** or **Menu** control.

Visual Basic: Windows Controls

Visual Studio 6.0

ItemCheck Event (ListView Control)

See Also Example Applies To

Occurs when the user checks an item.

Syntax

Private Sub *object*_**ItemCheck** ([*Index* **As Integer,**] **ByVal** *Item* **As ComctlLib.ListItem**)

The ItemCheck 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.
Item	Returns a reference to the clicked **ListItem** object.

Remarks

Checkboxes appear only when the **Checkboxes** property is set to **True**, and the **View** property is set to **lvwReport**.

Visual Basic Reference

Visual Studio 6.0

ItemReloaded Event

See Also Example Applies To

Occurs after a component is reloaded.

Syntax

Private Sub *object*_**ItemReloaded(***vbcomponent* **As VBComponent)**

The ItemReloaded event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
vbcomponent	A **VBComponent** object representing the component that was reloaded.

Visual Basic Extensibility Reference

Visual Studio 6.0

ItemRemoved Event (VBA Add-In Object Model)

See Also Example Applies To Specifics

Occurs after a reference is removed from a project.

Syntax

Sub *object*_**ItemRemoved(ByVal** *item* **As Reference)**

The required *item* argument specifies the **Reference** that was removed.

Visual Basic Extensibility Reference

Visual Studio 6.0

ItemRemoved Event (Add-Ins)

See Also Example Applies To

Occurs after a project, control, or component is removed from the current project.

Syntax

Sub *object*_**ItemRemoved (***vbcontrol* **As VBControl)**

Sub *object*_**ItemRemoved (***vbproject* **As VBProject)**

Sub *object*_**ItemRemoved (***vbcomponent* **As VBComponent)**

The ItemRemoved event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
vbproject	A **VBProject** object specifying the name of the project that was removed.
vbcomponent	A **VBComponent** object specifying the name of the component that was removed.
vbcontrol	A **VBControl** or **SelectedVBControl** object specifying the name of the component that was removed.

Visual Basic Extensibility Reference

Visual Studio 6.0

ItemRenamed Event

See Also Example Applies To

Occurs after a project, control, or component is renamed in the current project.

Syntax

Sub *object*_**ItemRenamed (***vbproject* **As VBProject)**

Sub *object*_**ItemRenamed (***vbcomponent* **As VBComponent)**

Sub *object*_**ItemRenamed (***vbcontrol* **As VBControl)**

The ItemRenamed event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
vbproject	A **VBProject** object specifying the name of the project that was renamed.
vbcomponent	A **VBComponent** object specifying the name of the component that was renamed.
vbcontrol	A **VBControl** object specifying the name of the control that was renamed.

Visual Basic Extensibility Reference

Visual Studio 6.0

ItemSelected Event

See Also Example Applies To

Occurs when a component in the Project window or an open designer-window is clicked.

Syntax

Sub *object*_**ItemSelected (***vbcomponent* **As VBComponent)**

The ItemSelected event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
vbcomponent	A **VBComponent** object specifying the name of the component that was selected.

Visual Basic Reference

Visual Studio 6.0

KeyDown, KeyUp Events (ActiveX Controls)

See Also Example Applies To

Occur when the user presses (KeyDown) or releases (KeyUp) a key while an object has the focus. (To interpret ANSI characters, use the KeyPress event.)

Syntax

Private Sub Form_KeyDown(*keycode* **As Integer**, *shift* **As Integer)**

Private Sub *object*_**KeyDown(**[*index* **As Integer**,]*keycode* **As Integer**, *shift* **As Integer)**

Private Sub Form_KeyUp(*keycode* **As Integer**, *shift* **As Integer)**

Private Sub *object*_**KeyUp(**[*index* **As Integer**,]*keycode* **As Integer**, *shift* **As Integer)**

The KeyDown and KeyUp event syntaxes have 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.
keycode	A key code, such as **vbKeyF1** (the F1 key) or **vbKeyHome** (the HOME key). To specify key codes, use the constants in the Visual Basic (VB) object library in the Object Browser.
shift	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event. The *shift* argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of *shift* is 6.

Remarks

For both events, the object with the focus receives all keystrokes. A form can have the focus only if it has no visible and enabled controls. Although the KeyDown and KeyUp events can apply to most keys, they're most often used for:

- Extended character keys such as function keys.
- Navigation keys.

- Combinations of keys with standard keyboard modifiers.
- Distinguishing between the numeric keypad and regular number keys.

Use KeyDown and KeyUp event procedures if you need to respond to both the pressing and releasing of a key.

KeyDown and KeyUp aren't invoked for:

- The ENTER key if the form has a **CommandButton** control with the **Default** property set to **True**.
- The ESC key if the form has a **CommandButton** control with the **Cancel** property set to **True**.
- The TAB key.

KeyDown and KeyUp interpret the uppercase and lowercase of each character by means of two arguments: *keycode*, which indicates the physical key (thus returning A and a as the same key) and *shift*, which indicates the state of *shift+key* and therefore returns either A or a.

If you need to test for the *shift* argument, you can use the *shift* constants which define the bits within the argument. The constants have the following values:

Constant	Value	Description
vbShiftMask	1	SHIFT key bit mask.
VbCtrlMask	2	CTRL key bit mask.
VbAltMask	4	ALT key bit mask.

The constants act as bit masks that you can use to test for any combination of keys.

You test for a condition by first assigning each result to a temporary integer variable and then comparing *shift* to a bit mask. Use the **And** operator with the *shift* argument to test whether the condition is greater than 0, indicating that the modifier was pressed, as in this example:

```
ShiftDown = (Shift And vbShiftMask) > 0
```

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

```
If ShiftDown And CtrlDown Then
```

Note If the **KeyPreview** property is set to **True**, a form receives these events before controls on the form receive the events. Use the **KeyPreview** property to create global keyboard-handling routines.

Visual Basic Reference

Visual Studio 6.0

KeyDown, KeyUp Events

See Also Example Applies To

Occur when the user presses (KeyDown) or releases (KeyUp) a key while an object has the focus. (To interpret ANSI characters, use the KeyPress event.)

Syntax

```
Private Sub Form_KeyDown(keycode As Integer, shift As Integer)

Private Sub object_KeyDown([index As Integer,]keycode As Integer, shift As Integer)

Private Sub Form_KeyUp(keycode As Integer, shift As Integer)

Private Sub object_KeyUp([index As Integer,]keycode As Integer, shift As Integer)
```

The KeyDown and KeyUp event syntaxes have 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.
keycode	A key code, such as **vbKeyF1** (the F1 key) or **vbKeyHome** (the HOME key). To specify key codes, use the constants in the Visual Basic (VB) object library in the Object Browser.
shift	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event. The *shift* argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of *shift* is 6.

Remarks

For both events, the object with the focus receives all keystrokes. A form can have the focus only if it has no visible and enabled controls. Although the KeyDown and KeyUp events can apply to most keys, they're most often used for:

- Extended character keys such as function keys.
- Navigation keys.

- Combinations of keys with standard keyboard modifiers.
- Distinguishing between the numeric keypad and regular number keys.

Use KeyDown and KeyUp event procedures if you need to respond to both the pressing and releasing of a key.

KeyDown and KeyUp aren't invoked for:

- The ENTER key if the form has a **CommandButton** control with the **Default** property set to **True**.
- The ESC key if the form has a **CommandButton** control with the **Cancel** property set to **True**.
- The TAB key.

KeyDown and KeyUp interpret the uppercase and lowercase of each character by means of two arguments: *keycode*, which indicates the physical key (thus returning A and a as the same key) and *shift*, which indicates the state of *shift+key* and therefore returns either A or a.

If you need to test for the *shift* argument, you can use the *shift* constants which define the bits within the argument. The constants have the following values:

Constant	Value	Description
vbShiftMask	1	SHIFT key bit mask.
VbCtrlMask	2	CTRL key bit mask.
VbAltMask	4	ALT key bit mask.

```
The constants act as bit masks that you can use to test for any combination of keys.
```

You test for a condition by first assigning each result to a temporary integer variable and then comparing *shift* to a bit mask. Use the **And** operator with the *shift* argument to test whether the condition is greater than 0, indicating that the modifier was pressed, as in this example:

```
ShiftDown = (Shift And vbShiftMask) > 0
```

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

```
If ShiftDown And CtrlDown Then
```

Note If the **KeyPreview** property is set to **True**, a form receives these events before controls on the form receive the events. Use the **KeyPreview** property to create global keyboard-handling routines.

Visual Basic Reference

KeyDown, KeyUp Events Example

This example demonstrates a generic keyboard handler that responds to the F2 key and to all the associated ALT, SHIFT, and CTRL key combinations. The key constants are listed in the Visual Basic (VB) object library in the Object Browser. To try this example, paste the code into the Declarations section of a form that contains a **TextBox** control, and then press F5 and press F2 with various combinations of the ALT, SHIFT, and CTRL keys.

```
Private Sub Text1_KeyDown (KeyCode As Integer, Shift As Integer)
   Dim ShiftDown, AltDown, CtrlDown, Txt
   ShiftDown = (Shift And vbShiftMask) > 0
   AltDown = (Shift And vbAltMask) > 0
   CtrlDown = (Shift And vbCtrlMask) > 0
   If KeyCode = vbKeyF2 Then    ' Display key combinations.
   If ShiftDown And CtrlDown And AltDown Then
      Txt = "SHIFT+CTRL+ALT+F2."
   ElseIf ShiftDown And AltDown Then
      Txt = "SHIFT+ALT+F2."
   ElseIf ShiftDown And CtrlDown Then
      Txt = "SHIFT+CTRL+F2."
   ElseIf CtrlDown And AltDown Then
      Txt = "CTRL+ALT+F2."
   ElseIf ShiftDown Then
      Txt = "SHIFT+F2."
   ElseIf CtrlDown Then
   Txt = "CTRL+F2."
   ElseIf AltDown Then
      Txt = "ALT+F2."
   ElseIf SHIFT = 0 Then
      Txt = "F2."
   End If
   Text1.Text = "You pressed " & Txt
   End If
End Sub
```

Visual Basic Reference

Visual Studio 6.0

KeyPress Event (ActiveX Controls)

See Also Example Applies To

Occurs when the user presses and releases an ANSI key.

Syntax

Private Sub Form_KeyPress(*keyascii* **As Integer)**

Private Sub *object*_**KeyPress(**[*index* **As Integer**,]*keyascii* **As Integer)**

The KeyPress 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.
keyascii	An integer that returns a standard numeric ANSI keycode. *Keyascii* is passed by reference; changing it sends a different character to the object. Changing *keyascii* to 0 cancels the keystroke so the object receives no character.

Remarks

The object with the focus receives the event. A form can receive the event only if it has no visible and enabled controls or if the **KeyPreview** property is set to **True**. A KeyPress event can involve any printable keyboard character, the CTRL key combined with a character from the standard alphabet or one of a few special characters, and the ENTER or BACKSPACE key. A KeyPress event procedure is useful for intercepting keystrokes entered in a **TextBox** or **ComboBox** control. It enables you to immediately test keystrokes for validity or to format characters as they're typed. Changing the value of the *keyascii* argument changes the character displayed.

You can convert the *keyascii* argument into a character by using the expression:

```
Chr(KeyAscii)
```

You can then perform string operations and translate the character back to an ANSI number that the control can interpret by using the expression:

```
KeyAscii = Asc(char)
```

Use KeyDown and KeyUp event procedures to handle any keystroke not recognized by KeyPress, such as function keys, editing keys, navigation keys, and any combinations of these with keyboard modifiers. Unlike the KeyDown and KeyUp events, KeyPress doesn't indicate the physical state of the keyboard; instead, it passes a character.

KeyPress interprets the uppercase and lowercase of each character as separate key codes and, therefore, as two separate characters. KeyDown and KeyUp interpret the uppercase and lowercase of each character by means of two arguments: *keycode*, which indicates the physical key (thus returning A and a as the same key), and *shift*, which indicates the state of *shift+key* and therefore returns either A or a.

If the **KeyPreview** property is set to **True**, a form receives the event before controls on the form receive the event. Use the **KeyPreview** property to create global keyboard-handling routines.

Note The ANSI number for the keyboard combination of CTRL+@ is 0. Because Visual Basic recognizes a *keyascii* value of 0 as a zero-length string (""), avoid using CTRL+@ in your applications.

Visual Basic Reference

Visual Studio 6.0

KeyPress Event

See Also Example Applies To

Occurs when the user presses and releases an ANSI key.

Syntax

```
Private Sub Form_KeyPress(keyascii As Integer)

Private Sub object_KeyPress([index As Integer,]keyascii As Integer)
```

The KeyPress 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.
keyascii	An integer that returns a standard numeric ANSI keycode. *Keyascii* is passed by reference; changing it sends a different character to the object. Changing *keyascii* to 0 cancels the keystroke so the object receives no character.

Remarks

The object with the focus receives the event. A form can receive the event only if it has no visible and enabled controls or if the **KeyPreview** property is set to **True**. A KeyPress event can involve any printable keyboard character, the CTRL key combined with a character from the standard alphabet or one of a few special characters, and the ENTER or BACKSPACE key. A KeyPress event procedure is useful for intercepting keystrokes entered in a **TextBox** or **ComboBox** control. It enables you to immediately test keystrokes for validity or to format characters as they're typed. Changing the value of the *keyascii* argument changes the character displayed.

You can convert the *keyascii* argument into a character by using the expression:

```
Chr(KeyAscii)
```

You can then perform string operations and translate the character back to an ANSI number that the control can interpret by using the expression:

```
KeyAscii = Asc(char)
```

Use KeyDown and KeyUp event procedures to handle any keystroke not recognized by KeyPress, such as function keys, editing keys, navigation keys, and any combinations of these with keyboard modifiers. Unlike the KeyDown and KeyUp events, KeyPress doesn't indicate the physical state of the keyboard; instead, it passes a character.

KeyPress interprets the uppercase and lowercase of each character as separate key codes and, therefore, as two separate characters. KeyDown and KeyUp interpret the uppercase and lowercase of each character by means of two arguments: *keycode*, which indicates the physical key (thus returning A and a as the same key), and *shift*, which indicates the state of *shift+key* and therefore returns either A or a.

If the **KeyPreview** property is set to **True**, a form receives the event before controls on the form receive the event. Use the **KeyPreview** property to create global keyboard-handling routines.

Note The ANSI number for the keyboard combination of CTRL+@ is 0. Because Visual Basic recognizes a *keyascii* value of 0 as a zero-length string (""), avoid using CTRL+@ in your applications.

Visual Basic Reference

KeyPress Event Example

This example converts text entered into a **TextBox** control to uppercase. To try this example, paste the code into the Declarations section of a form that contains a **TextBox**, and then press F5 and enter something into the **TextBox**.

```
Private Sub Text1_KeyPress (KeyAscii As Integer)
   Char = Chr(KeyAscii)
   KeyAscii = Asc(UCase(Char))
End Sub
```