

This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

Join Property

[See Also](#) [Example](#) [Applies To](#)

Returns or sets a value that determines how line segments are formed.

Syntax

object.**Join** [= *type*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>type</i>	Integer. A VtPenJoin constant that describes the style of pen join.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

KeepTogether Property

See Also Example [Applies To](#)

Returns or sets a value that specifies if the text of a section will be kept together (not broken up over a page break).

Syntax

object.**KeepTogether** [= *boolean*]

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

Part	Description
<i>object</i>	Required. An object expression that evaluates to an object in the Applies To list.
<i>boolean</i>	Optional. A boolean expression that specifies if a new page is begun, as shown in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	A page break occurs.
False	(Default) A page break doesn't occur.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

Key Property

See Also [Example](#) [Applies To](#)

Returns or sets a string that uniquely identifies a member in a [collection](#).

Syntax

object.**Key** [= *string*]

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

Part	Description
<i>object</i>	Required. An object expression that evaluates to an object in the Applies To list.
<i>string</i>	A unique string identifying a member in a collection.

Remarks

If the string is not unique, an error will occur.

You can set the **Key** property when you use the **Add** method to add an object to a collection.

If you expect the **Index** property to change dynamically, refer to objects in a collection using the **Key** property.

In addition, you can use the **Key** property to make your Visual Basic project "self-documenting" by assigning meaningful names to the objects in a collection.

© 2017 Microsoft

Visual Basic Reference

Add Method, ExportFormats Collection, Template Property Example

This example creates a template, adds an **ExportFormat** object to the **ExportFormats** collection using the new template, and exports the report using the **ExportFormat** object.

```
Private Sub ExportDailyReport()  
    DataReport1.Title = "Daily Report" ' This title appears in the report.  
    Dim strTemplate As String  
    ' Create the template.  
    strTemplate = _  
    "<HTML>" & vbCrLf & _  
    "<HEAD>" & vbCrLf & _  
    "<TITLE>" & "MyCompany: " & rptTagTitle & _  
    "</TITLE>" & vbCrLf & _  
    "<BODY>" & vbCrLf & _  
    rptTagBody & vbCrLf & _  
    "<BODY>" & vbCrLf & _  
    "</HTML>"  
  
    ' Add a new ExportFormat object using the template.  
    DataReport1.ExportFormats.Add _  
    Key:="DailyReport", _  
    FormatType:=rptFmtHTML, _  
    FileFormatString:="Daily Report (*.htm)", _  
    FileFilter:="*.HTM", _  
    Template:=strTemplate  
  
    ' Export the report using the new ExportFormat object.  
    DataReport1.ExportReport _  
    FormatIndexOrKey:="DailyReport", _  
    FileName:="C:\Temp\DailyRpt", _  
    Overwrite:=True, _  
    ShowDialog:=False, _  
    Range:=rptRangeFromTo, _  
    Pagefrom:=1, _  
    Pageto:=10  
End Sub
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

Key Property (ActiveX Controls)

See Also Example [Applies To](#)

Returns or sets a string that uniquely identifies a member in a [collection](#).

Syntax

object.**Key** [= *string*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>string</i>	A unique string identifying a member in a collection.

Remarks

If the string is not unique, an error will occur.

You can set the **Key** property when you use the **Add** method to add an object to a collection.

The value of the **Index** property of an object can change when objects in the collection are reordered, such as when you set the **Sorted** property to **True**. If you expect the **Index** property to change dynamically, refer to objects in a collection using the **Key** property.

In addition, you can use the **Key** property to make your Visual Basic project "self-documenting" by assigning meaningful names to the objects in a collection.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: RDO Data Control

Visual Studio 6.0

KeyColumn Property

[See Also](#) [Example](#) [Applies To](#)

Returns or sets a value that specifies if this column is part of the primary key.

Syntax

object.**KeyColumn** [= *value*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>value</i>	A Boolean expression as described in Settings.

Settings

The **KeyColumn** property has these settings:

Setting	Description
True	If the column is part of the primary key.
False	(Default) If the column is not part of the primary key.

Remarks

This property indicates if the column is part of the primary key for the result set. This property will be read/write when using the client batch cursor library (**CursorDriver** property set to **rdUseClientBatch**) and generates a trappable error when accessed using server-side cursors or ODBC cursor library.

When using the client batch cursor library, you can set this property to indicate which columns make up the primary key of the result set. This assists the cursor library when it builds the WHERE clauses for the update or delete/insert statements during an optimistic batch update.

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

KeyPreview Property

[See Also](#) [Example](#) [Applies To](#)

Returns or sets a value that determines whether keyboard events for forms are invoked before keyboard events for controls. The keyboard events are `KeyDown`, `KeyUp`, and `KeyPress`.

Syntax

object.**KeyPreview** [= *boolean*]

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

Part	Description
<i>Object</i>	An object expression that evaluates to an object in the Applies To list.
<i>Boolean</i>	A Boolean expression that specifies how events are received, as described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	The form receives keyboard events first and then the active control.
False	(Default) The active control receives keyboard events; the form doesn't.

Remarks

You can use this property to create a keyboard-handling procedure for a form. For example, when an application uses function keys, you'll want to process the keystrokes at the form level rather than writing code for each control that might receive keystroke events.

If a form has no visible and enabled controls, it automatically receives all keyboard events.

To handle keyboard events only at the form level and not allow controls to receive keyboard events, set *KeyAscii* to 0 in the form's `KeyPress` event, and set *KeyCode* to 0 in the form's `KeyDown` event.

Note Some controls intercept keyboard events so that the form can't receive them. Examples include the ENTER key when focus is on a **CommandButton** control and arrow keys when focus is on a **ListBox** control.

© 2017 Microsoft

Visual Basic Reference

KeyPreview Property Example

This example creates a form keyboard handler in the KeyDown event. Each of the first four function keys displays a different message. To try this example, paste the code into the Declarations section of a form, and then press F5. Once the program is running, press any one of the first four (F1 F4) function keys.

```
Private Sub Form_Load ()  
    KeyPreview = True  
End Sub  
  
Private Sub Form_KeyDown (KeyCode As Integer, Shift As Integer)  
    Select Case KeyCode  
        Case vbKeyF1: MsgBox "F1 is your friend."  
        Case vbKeyF2: MsgBox "F2 could copy text."  
        Case vbKeyF3: MsgBox "F3 could paste text."  
        Case vbKeyF4: MsgBox "F4 could format text."  
    End Select  
End Sub
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: RDO Data Control

Visual Studio 6.0

KeysetSize Property (Remote Data)

[See Also](#) [Example](#) [Applies To](#)

Returns or sets a value indicating the number of [rows](#) in the [keyset](#) buffer.

Syntax

object.**KeysetSize** [= *value*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>value</i>	A Long expression as described in Settings.

Settings

The settings for *value* must be greater than or equal to the **RowsetSize** property.

Remarks

The **KeysetSize** property is a value that specifies the number of rows in the keyset for a [keyset-](#) or [dynamic-type](#) **rdoResultset** [cursor](#). If the keyset size is 0 (the default), the cursor is fully keyset-driven. If the keyset size is greater than 0, the cursor is mixed (keyset-driven within the keyset and dynamic outside the keyset).

If **KeysetSize** is a value greater than **RowsetSize**, the value defines the number of rows in the keyset that are to be buffered by the driver.

Not all [ODBC data sources](#) support keyset cursors.

Note Because version 2.5 of the Microsoft SQL Server ODBC driver does not support mixed-style cursors, if you set a *value*, **KeysetSize** is reset to 0 and the driver returns error 01S02: "Option value changed."

Warning When using **rdConcurLock** concurrency (pessimistic), the **KeysetSize** determines the number of rows locked when the cursor is first opened. The entire keyset remains locked as long as the cursor remains open.

This documentation is archived and is not being maintained.

Visual Basic: Windows Controls

Visual Studio 6.0

LabelEdit Property

[See Also](#) [Example](#) [Applies To](#)

Returns or sets a value that determines if a user can edit labels of **ListItem** or **Node** objects in a **ListView** or **TreeView** control.

Syntax

```
object.LabelEdit [ = integer]
```

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>integer</i>	An integer that determines whether the label of a Node or ListItem object can be edited, as specified in Settings.

Settings

The settings for *integer* are:

Constant	Value	Description
ListView: lvwAutomatic TreeView: tvwAutomatic	0	(Default) Automatic. The BeforeLabelEdit event is generated when the user clicks the label of a selected node.
ListView: lvwManual TreeView: tvwManual	1	Manual. The BeforeLabelEdit event is generated only when the StartLabelEdit method is invoked.

Remarks

Label editing of an object is initiated when a selected object is clicked (if the **LabelEdit** property is set to Automatic). That is, the first click on an object will select it; a second (single) click on the object will initiate the label editing operation.

The **LabelEdit** property, in combination with the **StartLabelEdit** method, allows you to programmatically determine when and which labels can be edited. When the **LabelEdit** property is set to 1, no label can be edited unless the **StartLabelEdit** method is invoked. For example, the following code allows the user to edit a **Node** object's label by clicking a Command button:

```
Private Sub Command1_Click()  
    ' Determine if the right Node is selected.  
    If TreeView1.SelectedItem.Index = 1 Then  
        TreeView1.StartLabelEdit ' Let user begin editing.  
    End If  
End Sub
```

© 2017 Microsoft

Visual Basic: Windows Controls

LabelEdit Property Example

This example initiates label editing when you click the Command button. It allows a **Node** object to be edited unless it is a root **Node**. The **LabelEdit** property must be set to **Manual**. To try the example, place a **TreeView** control and a **CommandButton** on a form. Paste the code into the form's Declarations section. Run the example, select a node to edit, and press the **CommandButton**.

```
Private Sub Form_Load()  
    Dim nodX As Node  
    Dim i As Integer  
    TreeView1.LabelEdit = tvwManual    ' Set property to manual.  
    Set nodX = TreeView1.Nodes.Add(,,," Node 1")    ' Add first node.  
  
    For i = 1 to 5    ' Add 5 nodes.  
        Set nodX = TreeView1.Nodes.Add(i,tvwChild,"Node " & CStr(i + 1))  
    Next i  
  
    nodX.EnsureVisible    ' Show all nodes.  
End Sub  
  
Private Sub Command1_Click()  
    ' Invoke the StartLabelEdit method on the selected node,  
    ' which triggers the BeforeLabelEdit event.  
    TreeView1.StartLabelEdit  
End Sub  
  
Private Sub TreeView_BeforeLabelEdit (Cancel As Integer)  
    ' If the selected item is the root, then cancel the edit.  
    If TreeView1.SelectedItem Is TreeView1.SelectedItem.Root Then  
        Cancel = True  
    End If  
End Sub  
  
Private Sub TreeView_AfterLabelEdit _  
(Cancel As Integer, NewString As String)  
    ' Assume user has entered some text and pressed the ENTER key.  
    ' Any nonempty string will be valid.  
    If Len(NewString) = 0 Then  
        Cancel = True  
    End If  
End Sub
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

LabelLevelCount Property

See Also Example [Applies To](#)

Returns the number of levels of labels for a given axis. Not available at design time.

Syntax

object.**LabelLevelCount** [= *count*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>count</i>	Integer. An integer that describes the number of labels.



This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

Labels Property

[See Also](#) [Example](#) [Applies To](#)

Returns a reference to a **Labels** collection that describes the labels on a chart axis.

Syntax

object.**Labels**

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

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

LabelsInsidePlot Property

See Also [Example](#) [Applies To](#)

Returns or sets a value that determines whether to leave the axis labels at the normal location or move them with the axis to the new intersection point.

Syntax

object.**LabelsInsidePlot** [= *boolean*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>boolean</i>	A Boolean expression that specifies where to display the axis labels, as described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	(Default) The axis labels remain at the normal location.
False	The labels move inside the plot to the new intersection point.

Remarks

If this property is set, then the Intersection object's **Auto** property is automatically set to **False**.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

LabelTick Property

[See Also](#) [Example](#) [Applies To](#)

Returns or sets a value that indicates whether category axis labels are centered on an axis tick mark.

Syntax

object.**LabelTick** [= *boolean*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>boolean</i>	A Boolean expression that specifies whether the item is displayed, as described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	The labels are centered on a tick mark.
False	(Default) The labels are centered between two tick marks.

Remarks

If this property is set, the object's **Auto** property is automatically set to **False**.

This documentation is archived and is not being maintained.

Visual Basic: Windows Controls

Visual Studio 6.0

LabelWrap Property (ListView Control)

[See Also](#) [Example](#) [Applies To](#)

Returns or sets a value that determines whether or not labels are wrapped when a **ListView** control is in Icon view.

Syntax

object.LabelWrap [= *boolean*]

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

Part	Description
<i>object</i>	An object expression that evaluates to a ListView control.
<i>boolean</i>	A Boolean expression specifying if the labels wrap, as described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	(Default) The labels wrap.
False	The labels don't wrap.

Remarks

The length of the label is determined by setting the icon spacing in the Control Panel, Desktop option, in Windows NT. In Windows 95 or later, use the Appearance tab in the Display control panel.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

LargeChange, SmallChange Properties

[See Also](#) [Example](#) [Applies To](#)

- **LargeChange** Returns or sets the amount of change to the **Value** property setting in a scroll bar control (**HScrollBar** or **VScrollBar**) when the user clicks the area between the scroll box and scroll arrow.
- **SmallChange** Returns or sets the amount of change to the **Value** property setting in a scroll bar control when the user clicks a scroll arrow.

Syntax

object.**LargeChange** [= *number*]
object.**SmallChange** [= *number*]

The **LargeChange** and **SmallChange** property syntaxes have these parts:

Part	Description
<i>Object</i>	An object expression that evaluates to an object in the Applies To list.
<i>Number</i>	An integer that specifies the amount of change to the Value property.

Remarks

For both properties, you can specify an integer between 1 and 32,767, inclusive. By default, each property is set to 1.

The Microsoft Windows operating environment automatically sets proportional scrolling increments for scroll bars on **MDI Form** objects, **ComboBox** controls, and **ListBox** controls based on the amount of data in the object. For the **HScrollBar** and **VScrollBar** controls, however, you must specify these increments. Use **LargeChange** and **SmallChange** to set scrolling increments appropriate to how the scroll bar is being used.

Typically, you set **LargeChange** and **SmallChange** at design time. You can also reset them in code at [run time](#) when the scrolling increment must change dynamically.

Note You set the maximum and minimum ranges of the **HScrollBar** and **VScrollBar** controls with the **Max** and **Min** properties.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

LargeChange, SmallChange Properties (ActiveX Controls)

[See Also](#) [Example](#) [Applies To](#)

- The **LargeChange** property sets the number of ticks the slider will move when you press the PAGEUP or PAGEDOWN keys, or when you click the mouse to the left or right of the slider.
- The **SmallChange** property sets the number of ticks the slider will move when you press the left or right arrow keys.

Syntax

object.**LargeChange** = *number*

object.**SmallChange** = *number*

The **LargeChange** and **SmallChange** property syntaxes have these parts:

Part	Description
<i>object</i>	An object expression that evaluates to a Slider control.
<i>number</i>	A Long integer specifying how many ticks the slider moves.

Remarks

The default for the **LargeChange** property is 5. The default for the **SmallChange** property is 1.

© 2017 Microsoft

Visual Basic Reference

LargeChange, SmallChange Properties Example

This example uses a scroll bar to move a **PictureBox** control across the form. To try this example, paste the code into the Declarations section of a form that contains a small **PictureBox** control and an **HScrollBar** control, and then press F5 and click the scroll bar.

```
Private Sub Form_Load ()
    HScroll1.Max = 100    ' Set maximum value.
    HScroll1.LargeChange = 20    ' Cross in 5 clicks.
    HScroll1.SmallChange = 5    ' Cross in 20 clicks.
    Picture1.Left = 0    ' Start picture at left.
    Picture1.BackColor = QBColor(3)    ' Set color of picture box.
End Sub
Private Sub HScroll1_Change ()
    ' Move picture according to scroll bar.
    Picture1.Left = (HScroll1.Value / 100) * ScaleWidth
End Sub
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic for Applications Reference

Visual Studio 6.0

LastDLLError Property

[See Also](#) [Example](#) [Applies To](#) [Specifics](#)

Returns a system error code produced by a call to a [dynamic-link library](#) (DLL). Read-only.

Remarks

The **LastDLLError** [property](#) applies only to DLL calls made from Visual Basic code. When such a call is made, the called function usually returns a code indicating success or failure, and the **LastDLLError** property is filled. Check the documentation for the DLL's functions to determine the return values that indicate success or failure. Whenever the failure code is returned, the Visual Basic application should immediately check the **LastDLLError** property. No exception is raised when the **LastDLLError** property is set.

© 2017 Microsoft

Visual Basic for Applications Reference

LastDLLError Property Example

When pasted into a **UserForm** module, the following code causes an attempt to call a DLL function. The call fails because the argument that is passed in (a null pointer) generates an error, and in any event, SQL cant be cancelled if it isnt running. The code following the call checks the return of the call, and then prints at the **LastDLLError** property of the **Err** object to reveal the error code. On systems without DLLs, **LastDLLError** always returns zero.

```
Private Declare Function SQLCancel Lib "ODBC32.dll" _
    (ByVal hstmt As Long) As Integer

Private Sub UserForm_Click()
    Dim RetVal
    ' Call with invalid argument.
    RetVal = SQLCancel(myhandle&)
    ' Check for SQL error code.
    If RetVal = -2 Then
        'Display the information code.
        MsgBox "Error code is :" & Err.LastDllError
    End If
End Sub
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: RDO Data Control

Visual Studio 6.0

LastModified Property (Remote Data)

[See Also](#) [Example](#) [Applies To](#)

Returns a [bookmark](#) indicating the most recently added or changed [row](#).

Syntax

object.**LastModified**

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

Return Values

The return value for this property is a **Variant**(string) data type, as described in Remarks.

Remarks

After you use the **AddNew** method to add a new row, or edit an existing row using the **Update** method, the **LastModified** property returns a bookmark as a pointer to the row most recently modified providing the keyset supports additions. That is, if new rows are added to the keyset as well as the underlying database table(s), the **LastModified** property will point to this new row in the keyset.

To position the current row pointer to this row, set the **Bookmark** property of the (same) **rdoResultset** object to the **LastModified** property.

If there have been no modifications against this **rdoResultset**, then the **LastModified** property returns 0.

Not all types of **rdoResultsets** support additions to their keysets so the **LastModified** property might return 0 after a row has just been added. For example, while a ODBC cursor static keyset can be updated, its rowset cannot be added to. Inserts are performed on the database, but not to the keyset, so the **LastModified** property always returns 0 in this case.

Server-side keyset cursors support additions to the keyset so the **AddNew** method sets the **LastModified** property as does the **Update** method.

Client-side static cursors do not add new rows to the cursor's membership, thus the **LastModified** property value is undefined when using the **AddNew** method. However, it is defined after the **Update** method is used. Server-side static cursors are read-only, so the **LastModified** property is not relevant in this case.

Dynamic and forward-only cursors do not support bookmarks (as indicated by the **Bookmarkable** property returning False), so the **LastModified** property is not relevant in these cursors.

The client batch cursor library also supports the **LastModified** property. For static cursors, new rows are added to the cursor membership so the **AddNew** method sets the **LastModified** property as does the **Update** method.

Visual Basic: RDO Data Control

LastModified, Bookmark Properties Example

This example illustrates use of the **LastModified** property to reposition the current row pointer to the row most recently modified by RDO. The code opens a connection against SQL Server and creates a keyset cursor-based query on the Authors table. The query expects a single parameter to pass in the name of the Author to edit. Once selected, edited and updated, the row pointer is repositioned to the last row modified by setting the bookmark property of the **rdoResultset** to the **LastModified** property.

```
Option Explicit
Dim er As rdoError
Dim cn As New rdoConnection
Dim qy As New rdoQuery
Dim rs As rdoResultset
Dim col As rdoColumn

Private Sub TestLM_Click()
    qy(0) = LookFor.Text

    rs.Edit
    rs!City = NewCity.Text ' a TextBox control
    rs.Update

    rs.Bookmark = rs.LastModified

    'Simply show data in picture control
    Pic.Cls 'Clear the picture control.

    For Each col In rs.rdoColumns
        Pic.Print col.Name,
    Next
    Pic.Print String(80, "-")
    For Each col In rs.rdoColumns
        Pic.Print col,
    Next

End Sub

Private Sub Form_Load()
    cn.CursorDriver = rdUseOdbc
    cn.Connect = "uid=;pwd=;server=sequel;" _
        & "driver={SQL Server};database=pubs;dsn='';"
    cn.EstablishConnection

    With qy
        .Name = "ShowWhite"
        .SQL = "Select * from Authors " _
            & " where Au_LName like ?"
        .LockType = rdConcurReadOnly
        .CursorType = rdOpenForwardOnly
        .RowsetSize = 1
        Set .ActiveConnection = cn
    End With
```

```
qy(0) = LookFor.Text      ' a textbox control  
Set rs = qy.OpenResultset(rdOpenKeyset, rdConcurRowver)
```

```
Exit Sub  
End Sub
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: RDO Data Control

Visual Studio 6.0

LastQueryResults Property

[See Also](#) [Example](#) [Applies To](#)

Returns a reference to the **rdoResultset** object generated by the last query if any.

Syntax

object.**LastQueryResults**

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

Return Values

The **LastQueryResults** property return value is an object expression that specifies a valid **rdoResultset** or Nothing if there is no result set available.

Remarks

This new property will contain a reference to the **rdoResultset** object generated by the last query executed on this connection, if any. The necessity of this property comes from the Queries as Methods feature, allowing the developer to call their queries and stored procedures as methods of the parent connection object. Since stored procedures can pass back a return value as well as resultsets, the developer needs this property in order to get a reference to the result set created during the last query as method call. The developer would use this property as shown below:

```
Dim RetCode As Long
Dim rs as rdoResultset
RetCode = MyConnection.MyQuery(x,y,z)
Set rs = MyConnection.LastQueryResults
```

This property is set back to Nothing on the next query execution on the connection. This property will be set for any query executed on the connection, even if the developer used the OpenResultset or Execute method instead of calling the query as a method of the connection object.

Note The value in **LastQueryResults** doesn't persist in RDO. It only temporarily holds the last resultset created until you check it, and then it is cleared until a new resultset is generated. As a result, if you check **LastQueryResults** again, it returns null.

If you want to use the value of **LastQueryResults** in a global variable, you should first assign it to a temporary variable and check to see if it equals **Nothing** before overwriting the global variable to avoid any problems. For example:

```
Dim trs As rdoResultset
cn.MyQuery
trs = cn.LastQueryResult
If trs Is Nothing Then
    Debug.print "No result"
Else
```

```
Set rs = trs
Debug.Print rs(0)
End If
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: Windows Controls

Visual Studio 6.0

LastSibling Property

[See Also](#) [Example](#) [Applies To](#)

Returns a reference to the last sibling of a **Node** object in a **TreeView** control.

Syntax

object.LastSibling

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

Remarks

The last sibling is the **Node** that appears in the last position in one level of a hierarchy of nodes. Which **Node** actually appears in the last position depends on whether or not the **Node** objects at that level are sorted, which is determined by the **Sorted** property. To sort the **Node** objects at one level, set the **Sorted** property of the **Parent** node to **True**. The following code demonstrates this:

```
Private Sub TreeView1_NodeClick(ByVal Node As Node)
    Node.Parent.Sorted = True
End Sub
```

The **Child**, **FirstSibling**, **LastSibling**, **Previous**, **Parent**, **Next**, and **Root** properties all return a reference to another **Node** object. Therefore, you can simultaneously reference and perform operations on a **Node**, as follows:

```
With TreeView1.Nodes(x).LastSibling
    .Text = "New text"
    .Key = "New key"
    .SelectedImage = 3
End With
```

You can also set an object variable to the referenced **Node**, as follows:

```
Dim NodLastSib As Node
' Get a reference to the last sibling of Node x.
Set NodLastSib = TreeView1.Nodes(x).LastSibling
' Use this reference to perform operations on the sibling Node.
With NodLastSib
    .Text = "New text"    ' Change the text.
    .Key = "New key"     ' Change key.
    .SelectedImage = 3   ' Change SelectedImage.
End With
```

© 2017 Microsoft

Visual Basic: Windows Controls

LastSibling Property Example

This example adds several **Node** objects to a **TreeView** control. The **LastSibling** property, in conjunction with the **Next** property and the **FirstSibling** property, is used to navigate through a clicked **Node** object's hierarchy level. To try the example, place a **TreeView** control on a form and paste the code into the form's Declarations section. Run the example, and click the various nodes to see what is returned.

```
Private Sub Form_Load()
    Dim nodX As Node
    Set nodX = TreeView1.Nodes.Add(,,"dad","Mike")
    Set nodX = TreeView1.Nodes.Add(,,"mom","Carol")
    ' Alice is the LastSibling.
    Set nodX = TreeView1.Nodes.Add(,,"Alice")

    Set nodX = TreeView1.Nodes.Add("mom",tvwChild,,"Marsha")
    Set nodX = TreeView1.Nodes.Add("mom",tvwChild,,"Jan")
    ' Cindy is the LastSibling.
    Set nodX = TreeView1.Nodes.Add("mom",tvwChild,,"Cindy")
    nodX.EnsureVisible ' Show all nodes.

    Set nodX = TreeView1.Nodes.Add("dad",tvwChild,,"Greg")
    Set nodX = TreeView1.Nodes.Add("dad",tvwChild,,"Peter")
    ' Bobby is the LastSibling.
    Set nodX = TreeView1.Nodes.Add("dad",tvwChild,,"Bobby")
    nodX.EnsureVisible ' Show all nodes.
End Sub

Private Sub TreeView1_NodeClick(ByVal Node As Node)
    Dim strText As String
    Dim n As Integer
    ' Set n to FirstSibling's index.
    n = Node.FirstSibling.Index
    ' Place FirstSibling's text & linefeed in string variable.
    strText = Node.FirstSibling.Text & vbCrLf
    While n <> Node.LastSibling.Index
        ' While n is not the index of the last sibling, go to the
        ' next sibling and place its text into the string variable
        strText = strText & TreeView1.Nodes(n).Next.Text & vbCrLf
        ' Set n to the next node's index.
        n = TreeView1.Nodes(n).Next.Index
    Wend
    MsgBox strText ' Display results.
End Sub
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Extensibility Reference

Visual Studio 6.0

LastUsedPath Property

See Also Example [Applies To](#)

Returns or sets the last path used for the file dialog boxes used in Visual Basic, such as the **Open Project** dialog box.

Syntax

object.**LastUsedPath** ([*pathname* **As String**])

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

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>pathname</i>	(Optional) A string containing the last path name used for a file dialog box.

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

LBound Property

[See Also](#) [Example](#) [Applies To](#)

Returns the lowest ordinal value of a control in a control array.

Syntax

object.**LBound**

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

Remarks

The **LBound** property setting is equal to the **Index** property value of the first control in the array. Typically this value is 0 because Visual Basic automatically assigns an **Index** value of 0 to the first control in a control array. If you manually change the **Index** value for the first control in an array to some other value (for example, 1), **LBound** returns the value you manually assigned to **Index** (in this example, 1).

© 2017 Microsoft

Visual Basic Reference

LBound, UBound Properties Example

This example prints the values of these two properties for a control array. Put an **OptionButton** control on a form, and set its **Index** property to 0 (to create a control array). To try this example, paste the code into the Declarations section of a form, and then press F5 and click the form.

```
Private Sub Form_Paint ()
    Static FlagFormPainted As Integer
    If FlagFormPainted <> True Then ' When form is painting for first time,
        For i = 1 To 3
            Load Option1(i) ' add three option buttons to array.
            Option1(i).Top = Option1(i - 1).Top + 350
            Option1(i).Visible = True
        Next I
        For I = 0 to 3 ' Put captions on the option buttons.
            Option1(i).Caption = "Option #" & CStr(i)
        Next I
        Option1(0).Value = True ' Select first option button.
        FlagFormPainted = True ' Form is done painting.
    End If
End Sub

Private Sub Form_Click ()
    Print "Control array's Count property is " & Option1().Count
    Print "Control array's LBound property is " & Option1().LBound
    Print "Control array's UBound property is " & Option1().UBound
End Sub
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Extensibility Reference

Visual Studio 6.0

Left Property

[See Also](#) [Example](#) [Applies To](#) [Specifics](#)

Returns or sets a Single containing the location of the left edge of the window on the screen in twips. Read/write.

Remarks

The value returned by the **Left** property depends on whether or not the window is linked or docked.

Note Changing the **Left** property setting of a linked or docked window has no effect as long as the window remains linked or docked.

© 2017 Microsoft

Visual Basic Extensibility Reference

Left, Top Properties Example

The following example uses the **Left** and **Top** properties to return the coordinates of the upper-left corner of a particular window, in twips. These property settings change after a window is linked or docked because then they refer to the **Window** object to which the original window is linked or docked.

```
Debug.Print Application.VBE.Windows(9).Left  
Debug.Print Application.VBE.Windows(9).Top
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

Left, Top Properties

[See Also](#) [Example](#) [Applies To](#)

- **Left** returns or sets the distance between the internal left edge of an object and the left edge of its container.
- **Top** returns or sets the distance between the internal top edge of an object and the top edge of its container.

Syntax

object.**Left** [= *value*]

object.**Top** [= *value*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>value</i>	A numeric expression specifying distance.

Remarks

For a form, the **Left** and **Top** properties are always expressed in twips; for a control, they are measured in units depending on the coordinate system of its container. The values for these properties change as the object is moved by the user or by code. For the **CommonDialog** and **Timer** controls, these properties aren't available at [run time](#).

For either property, you can specify a single-precision number.

Use the **Left**, **Top**, **Height**, and **Width** properties for operations based on an object's external dimensions, such as moving or resizing. Use the **ScaleLeft**, **ScaleTop**, **ScaleHeight**, and **ScaleWidth** properties for operations based on an object's internal dimensions, such as drawing or moving objects that are contained within the object. The scale-related properties apply only to **PictureBox** controls and **Form** and **Printer** objects.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

Left, Top Properties (ActiveX Controls)

[See Also](#) [Example](#) [Applies To](#)

- **Left** returns or sets the distance between the internal left edge of an object and the left edge of its container.
- **Top** returns or sets the distance between the internal top edge of an object and the top edge of its container.
- For the **Panel** object only, this is a read-only property.

Syntax

object.**Left** [= *value*]

object.**Top** [= *value*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>value</i>	A numeric expression specifying distance.

Remarks

For a form, the **Left** and **Top** properties are always expressed in twips; for a control, they are measured in units depending on the coordinate system of its container. The values for these properties change as the object is moved by the user or by code. For the **CommonDialog** and **Timer** controls, these properties aren't available at [run time](#).

For either property, you can specify a single-precision number.

Use the **Left**, **Top**, **Height**, and **Width** properties for operations based on an object's external dimensions, such as moving or resizing. Use the **ScaleLeft**, **ScaleTop**, **ScaleHeight**, and **ScaleWidth** properties for operations based on an object's internal dimensions, such as drawing or moving objects that are contained within the object. The scale-related properties apply only to **PictureBox** controls and **Form** and **Printer** objects.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: DataGrid Control

Visual Studio 6.0

LeftCol Property

See Also Example [Applies To](#)

Returns or sets an integer representing the leftmost visible column of a **DataGrid** control. this property is read-only at design time.

Syntax

object.**LeftCol** [= *value*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>value</i>	A numeric expression indicating the leftmost visible column. The default value is 0.

This documentation is archived and is not being maintained.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

LeftCol Property (MSHFlexGrid)

[See Also](#) [Example](#) [Applies To](#)

Returns or sets the left-most visible non fixed column in the **MSHFlexGrid**. This property is not available at design time.

Syntax

object.**LeftCol** [= *value*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>value</i>	An integer that specifies the left-most column.

Remarks

This property can be used programmatically to scroll within the **MSHFlexGrid**. Use the **TopRow** property to determine the topmost visible row of the **MSHFlexGrid**.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

LeftMargin, RightMargin Properties

See Also Example [Applies To](#)

Returns or sets, in twips, the width of the left or right margin.

Syntax

object.**LeftMargin** [=*number*]

object.**RightMargin**[=*number*]

The **LeftMargin** and **RightMargin** properties syntax have these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>number</i>	Optional. A numeric expression specifying the width of the left or right margin.

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

LegalCopyright Property

[See Also](#) [Example](#) [Applies To](#)

Returns or sets a string value containing legal copyright information about the running application. Read only at [run time](#).

Syntax

object.**LegalCopyright**

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

Remarks

You can set this property at design time in the Type box in the Make tab of the Project Properties dialog box.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

LegalTrademarks Property

[See Also](#) [Example](#) [Applies To](#)

Returns or sets a string value containing legal trademark information about the running application. Read only at [run time](#).

Syntax

object.**LegalTrademarks**

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

Remarks

You can set this property at design time in the Type box in the Make tab of the Project Properties dialog box.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

Legend Property

See Also [Example](#) [Applies To](#)

Returns a reference to a **Legend** object that contains information about the appearance and behavior of the graphical key and accompanying text that describes the chart series.

Syntax

object.**Legend**

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

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

LegendText Property

See Also Example [Applies To](#)

Returns or sets the text that identifies the series in the legend of a chart.

Syntax

object.**LegendText** [= *text*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>text</i>	String. The text that identifies the current series in the legend.

Remarks

By default, this text is the same as the **Text** property of the **ColumnLabel** object.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

Length Property (MSChart)

See Also Example [Applies To](#)

Returns or sets the length of axis tick marks, measured in points.

Syntax

object.Length [= *length*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>length</i>	Integer. The axis tick mark length.

This documentation is archived and is not being maintained.

Visual Basic: Multimedia MCI Control

Visual Studio 6.0

Length Property (Multimedia MCI Control)

[See Also](#) [Example](#) [Applies To](#)

Specifies the length, as defined by the **Multimedia MCI** control **TimeFormat** property, of the media in an open MCI device. This property is not available at design time and is read-only at run time.

Syntax

`[form.]MMControl.Length`

Data Type

Long

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

LicenseKey Property

See Also Example [Applies To](#)

Returns the license key of a control.

Syntax

object.**LicenseKey**

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

Remarks

You must add the license key to the **Licenses** collection before attempting to dynamically add a licensed control to the **Controls** collection.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

Licenses Property

[See Also](#) [Example](#) [Applies To](#)

Returns a reference to the **Licenses** Collection.

Syntax

object.**Licenses**

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

Remarks

The **Licenses** collection is used to contain license keys of unreferenced controls. You must add a control's license key to the collection before adding the control itself to the **Controls** collection. To add a license key, use the **Add** method for the **Licenses** collection.

The **Licenses** property is a property of the **Global** object. Since the **Global** object is referenced automatically, you can use the **Licenses** property without explicitly naming the object, as shown in the code below:

```
Licenses.Add "myProject.myControl", "xydsfasfjewfe"
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

Light Property

See Also Example [Applies To](#)

Returns a reference to a **Light** object that provides information about the light illuminating a three-dimensional chart.

Syntax

object.**Light**

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

© 2017 Microsoft



This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

LightSources Property

[See Also](#) [Example](#) [Applies To](#)

Returns a reference to a **LightSources** collection that describes all light sources used to illuminate a three-dimensional chart.

Syntax

object.**LightSources**

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

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

Limit Property

See Also [Example](#) [Applies To](#)

Returns or sets the joint limit, in points, of the line.

Syntax

object.**Limit** [= *joint*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>joint</i>	Single. A joint limit as a multiple of the line width. If two lines meet at a sharp angle, a mitered join results in a corner point that extends beyond the actual corner. If the distance from the inner join point to the outer join point exceeds the value in this variable, the join automatically changes to a bevel.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic for Applications Reference

Visual Studio 6.0

Line Property

[See Also](#) [Example](#) [Applies To](#) [Specifics](#)

Description

Read-only property that returns the current line number in a **TextStream** file.

Syntax

object.**Line**

The *object* is always the name of a **TextStream** object.

Remarks

After a file is initially opened and before anything is written, **Line** is equal to 1.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

LineSlant Property

See Also Example [Applies To](#)

Returns or sets the direction the line will slant.

Syntax

object.**LineSlant** [= *integer*]

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

Part	Description
<i>object</i>	Required. An object expression that evaluates to an object in the Applies To list.
<i>integer</i>	Optional. A numeric expression that specifies the direction of the line's slant, as shown in Settings.

Settings

The settings for *integer* are:

Constant	Value	Description
rptSlantNWSE	0	The line slants from upper left to lower right (northwest to southeast).
rptSlantNESW	1	The line slants from upper right to lower left (northeast to southwest).

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

LineStyle Property (DataPointLabel Object)

[See Also](#) [Example](#) [Applies To](#)

Returns or sets the type of line used to connect a data point to a label on a chart.

Syntax

object.**LineStyle** [= *type*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>type</i>	Integer. A VtChLabelLineStyle constant identifying the connecting line.

This documentation is archived and is not being maintained.

Visual Basic: Windows Controls

Visual Studio 6.0

LineStyle Property (TreeView Control)

[See Also](#) [Example](#) [Applies To](#)

Returns or sets the style of lines displayed between **Node** objects.

Syntax

object.**LineStyle** [= *number*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>number</i>	A value or constant that specifies the line style as shown in Settings.

Settings

The settings for *number* are:

Constant	Value	Description
tvwTreeLines	0	(Default) Tree lines. Displays lines between Node siblings and their parent Node .
tvwRootLines	1	Root Lines. In addition to displaying lines between Node siblings and their parent Node , also displays lines between the root nodes.

Remarks

You must set the **Style** property to a style that includes tree lines.

© 2017 Microsoft

Visual Basic: Windows Controls

LineStyle, Style Properties Example

This example adds several **Node** objects with images to a **TreeView** control. You can change the **LineStyle** and **Style** properties by selecting the alternate styles in two **OptionButton** control arrays. To try the example, place a **TreeView** control, an **ImageList** control, and two **OptionButton** control arrays (one with two buttons and one with eight) on a form, and paste the code into the form's Declarations section. Run the example, and click any **OptionButton** to change the **LineStyle** and **Style** properties.

```
Private Sub Form_Load()
    ' Add an image to the ImageList control.
    Dim imgX As ListImage
    Set imgX = ImageList1.ListImages. _
    Add(, , LoadPicture("bitmaps\outline\leaf.bmp"))

    TreeView1.ImageList = ImageList1 ' Initialize ImageList.

    ' Label OptionButton controls with line styles choices.
    Option1(0).Caption = "TreeLines"
    Option1(1).Caption = "RootLines"

    ' Select the first option, and set the LineStyle to TreeLines initially
    Option1(0).Value = True
    Treeview1.LineStyle = tvwTreeLines

    ' Label OptionButton controls with Style choices.
    Option2(0).Caption = "Text only"
    Option2(1).Caption = "Image & text"
    Option2(2).Caption = "Plus/minus & text"
    Option2(3).Caption = "Plus/minus, image & text"
    Option2(4).Caption = "Lines & text"
    Option2(5).Caption = "Lines, image & Text"
    Option2(6).Caption = "Lines, plus/minus & Text"
    Option2(7).Caption = "Lines, plus/minus, image & text"

    ' Select the last option, and set the initial Style
    Option2(7).Value = True
    Treeview1.Style = tvwTreelinesPlusMinusPictureText

    Dim nodX As Node
    Dim i as Integer
    ' Create root node.
    Set nodX = TreeView1.Nodes.Add(, , "Node " & "1", 1)

    For i = 1 to 5 ' Add 5 nodes.
        Set nodX = TreeView1.Nodes. _
        Add(i, tvwChild, "Node " & CStr(i + 1), 1)
    Next i
    nodX.EnsureVisible ' Show all nodes.
End Sub

Private Sub Option1_Click(Index as Integer)
    ' Change line style from OptionButton.
    TreeView1.LineStyle = Index
End Sub
```


End Sub

```
Private Sub Option2_Click(Index as Integer)
' Change Style with OptionButton.
    TreeView1.Style = Index
    Form1.Caption = "TreeView Style = " & Option2(Index).Caption
End Sub
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Extensibility Reference

Visual Studio 6.0

LinkedWindowFrame Property

[See Also](#) [Example](#) [Applies To](#) [Specifics](#)

Returns the **Window** object representing the frame that contains the window. Read-only.

Remarks

The **LinkedWindowFrame** property enables you to access the object representing the linked window frame, which has properties distinct from the window or windows it contains. If the window isn't linked, the **LinkedWindowFrame** property returns **Nothing**.

© 2017 Microsoft



This documentation is archived and is not being maintained.

Visual Basic Extensibility Reference

Visual Studio 6.0

LinkedWindows Property

See Also Example [Applies To](#) Specifics

Returns the collection of all linked windows contained in a linked window frame. Read-only.

Remarks

The **LinkedWindows** property is an accessor property (that is, a property that returns an object of the same type as the property name).

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

LinkItem Property

[See Also](#) [Example](#) [Applies To](#)

Returns or sets the data passed to a destination control in a DDE conversation with another application.

Syntax

object.**LinkItem** [= *string*]

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

Part	Description
<i>Object</i>	An object expression that evaluates to an object in the Applies To list.
<i>String</i>	A string expression that specifies the data to be passed to the destination control.

Remarks

This property corresponds to the *item* argument in the standard DDE syntax, with *application*, *topic*, and *item* as arguments. To set this property, specify a recognizable unit of data in an application as a reference for example, a cell reference such as "R1C1" in Microsoft Excel.

Use **LinkItem** in combination with the **LinkTopic** property to specify the complete data link for a destination control to a source application. To activate this link, set the **LinkMode** property.

You set **LinkItem** only for a control used as a destination. When a Visual Basic form is a source in a DDE conversation, the name of any **Label**, **PictureBox**, or **TextBox** control on the form can be the *item* argument in the *application|topic!item* string used by the destination. For example, the following syntax represents a valid reference from Microsoft Excel to a Visual Basic application:

`=VizBasicApplication|MyForm!TextBox1`

You could enter the preceding syntax for a destination cell in the Microsoft Excel formula bar.

A DDE control can potentially act as destination and source simultaneously, causing an infinite loop if a destination-source pair is also a source-destination pair with itself. For instance, a **TextBox** control may be both a source (through its parent form) and destination of the same cell in Microsoft Excel. When data in a Visual Basic **TextBox** changes, sending data to Microsoft Excel, the cell in Microsoft Excel changes, sending the change to the **TextBox**, and so on, causing the loop.

To avoid such loops, use related but not identical items for destination-source and source-destination links in both directions between applications. For example, in Microsoft Excel, use related cells (precedents or dependents) to link a worksheet with a

Visual Basic control, avoiding use of a single item as both destination and source. Document any *application|topic* pairs you establish if you include a Paste Link command for [run-time](#) use.

Note Setting a permanent data link at design time with the Paste Link command from the Edit menu also sets the **LinkMode**, **LinkTopic**, and **LinkItem** properties. This creates a link that is saved with the form. Each time the form is loaded, Visual Basic attempts to re-establish the conversation.

© 2017 Microsoft

Visual Basic Reference

LinkItem, LinkMode, LinkTopic Properties Example

In the example, each mouse click causes a cell in a Microsoft Excel worksheet to update the contents of a Visual Basic **TextBox** control. To try this example, start Microsoft Excel, open a new worksheet named Sheet1, and put some data in the first column. In Visual Basic, create a form with a **TextBox** control. Paste the code into the Declarations section, and then press F5 to run the program.

```
Private Sub Form_Click ()
    Dim CurRow As String
    Static Row    ' Worksheet row number.
    Row = Row + 1  ' Increment Row.
    If Row = 1 Then  ' First time only.
        ' Make sure the link isn't active.
        Text1.LinkMode = 0
        ' Set the application name and topic name.
        Text1.LinkTopic = "Excel|Sheet1"
        Text1.LinkItem = "R1C1"    ' Set LinkItem.
        Text1.LinkMode = 1    ' Set LinkMode to Automatic.
    Else
        ' Update the row in the data item.
        CurRow = "R" & Row & "C1"
        Text1.LinkItem = CurRow    ' Set LinkItem.
    End If
End Sub
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

LinkMode Property

[See Also](#) [Example](#) [Applies To](#)

Returns or sets the type of link used for a DDEconversation and activates the connection as follows:

- **Control** Allows a destination control on a Visual Basic form to initiate a conversation, as specified by the control's **LinkTopic** and **LinkItem** properties.
- **Form** Allows a destination application to initiate a conversation with a Visual Basic source form, as specified by the destination application's *application|topic|item* expression.

Syntax

object.**LinkMode** [= *number*]

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

Part	Description
<i>Object</i>	An object expression that evaluates to an object in the Applies To list.
<i>Number</i>	An integer that specifies the type of connection, as described in Settings.

Settings

For controls used as destinations in DDE conversations, the settings for *number* are:

Constant	Setting	Description
VbLinkNone	0	(Default) None No DDE interaction.
VbLinkAutomatic	1	Automatic Destination control is updated each time the linked data changes.
VbLinkManual	2	Manual {Destination control is updated only when the LinkRequest method is invoked.
VbLinkNotify	3	Notify A LinkNotify event occurs whenever the linked data changes, but the destination control is updated only when the LinkRequest method is invoked.

For forms used as sources in DDE conversations, the settings for *number* are:

Constant	Setting	Description
VbLinkNone	0	(Default) None No DDE interaction. No destination application can initiate a conversation with the source form as the topic, and no application can poke data to the form. If LinkMode is 0 (None) at design time, you can't change it to 1 (Source) at run time .
VbLinkSource	1	Source Allows any Label , PictureBox , or TextBox control on a form to supply data to any destination application that establishes a DDE conversation with the form. If such a link exists, Visual Basic automatically notifies the destination whenever the contents of a control are changed. In addition, a destination application can poke data to any Label , PictureBox , or TextBox control on the form. If LinkMode is 1 (Source) at design time, you can change it to 0 (None) and back at run time.

Remarks

The following conditions also apply to the **LinkMode** property:

- Setting **LinkMode** to a nonzero value for a destination control causes Visual Basic to attempt to initiate the conversation specified in the **LinkTopic** and **LinkItem** properties. The source updates the destination control according to the type of link specified (automatic, manual, or notify).
- If a source application terminates a conversation with a Visual Basic destination control, the value for that control's **LinkMode** setting changes to 0 (None).
- If you leave **LinkMode** for a form set to the default 0 (None) at design time, you can't change **LinkMode** at run time. If you want a form to act as a source, you must set **LinkMode** to 1 (Source) at design time. You can then change the value of **LinkMode** at run time.

Note Setting a permanent data link at design time with the Paste Link command from the Edit menu also sets the **LinkMode**, **LinkTopic**, and **LinkItem** properties. This creates a link that is saved with the form. Each time the form is loaded, Visual Basic attempts to re-establish the conversation.

© 2017 Microsoft

Visual Basic Reference

LinkItem, LinkMode, LinkTopic Properties Example

In the example, each mouse click causes a cell in a Microsoft Excel worksheet to update the contents of a Visual Basic **TextBox** control. To try this example, start Microsoft Excel, open a new worksheet named Sheet1, and put some data in the first column. In Visual Basic, create a form with a **TextBox** control. Paste the code into the Declarations section, and then press F5 to run the program.

```
Private Sub Form_Click ()
    Dim CurRow As String
    Static Row    ' Worksheet row number.
    Row = Row + 1 ' Increment Row.
    If Row = 1 Then ' First time only.
        ' Make sure the link isn't active.
        Text1.LinkMode = 0
        ' Set the application name and topic name.
        Text1.LinkTopic = "Excel|Sheet1"
        Text1.LinkItem = "R1C1" ' Set LinkItem.
        Text1.LinkMode = 1 ' Set LinkMode to Automatic.
    Else
        ' Update the row in the data item.
        CurRow = "R" & Row & "C1"
        Text1.LinkItem = CurRow ' Set LinkItem.
    End If
End Sub
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

LinkTimeout Property

[See Also](#) [Example](#) [Applies To](#)

Returns or sets the amount of time a control waits for a response to a DDE message.

Syntax

object.**LinkTimeout** [= *number*]

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

Part	Description
<i>Object</i>	An object expression that evaluates to an object in the Applies To list.
<i>Number</i>	A numeric expression that specifies the wait time.

Remarks

By default, the **LinkTimeout** property is set to 50 (equivalent to 5 seconds). You can specify other settings in tenths of a second.

DDE response time from source applications varies. Use this property to adjust the time a destination control waits for a response from a source application. If you use **LinkTimeout**, you can avoid generating a Visual Basic error if a given source application takes too long to respond.

Note The maximum length of time that a control can wait is 65,535 tenths of a second, or about 1 hour 49 minutes. Setting **LinkTimeout** to 1 tells the control to wait the maximum length of time for a response in a DDE conversation. The user can force the control to stop waiting by pressing the ESC key.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

LinkTopic Property

[See Also](#) [Example](#) [Applies To](#)

For a destination control returns or sets the source application and the topic (the fundamental data grouping used in that application). Use **LinkTopic** with the **LinkItem** property to specify the complete data link.

For a source form returns or sets the topic that the source form responds to in a DDE conversation.

Syntax

`object.LinkTopic [= value]`

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>value</i>	A string expression specifying a DDE syntax element.

Remarks

The **LinkTopic** property consists of a string that supplies part of the information necessary to set up either a destination link or source link. The string you use depends on whether you're working with a destination control or a source form. Each string corresponds to one or more elements of standard DDE syntax, which include *application*, *topic*, and *item*.

Note While the standard definition for a DDE link includes the *application*, *topic*, and *item* elements, the actual syntax used within applications for a destination link to a source application may vary slightly. For example, within Microsoft Excel, you use the syntax:

`application|topic!item`

Within Microsoft Word for Windows, you use:

`application topic item`

(Don't use the pipe character [|] or exclamation mark [!].)

Within a Visual Basic application, you use:

`application|topic`

The exclamation mark for *topic* is implicit.

Destination Control To set **LinkTopic** for a destination control, use a string with the syntax *application|topic* as follows:

- *application* is the name of the application from which data is requested, usually the executable filename without an extension for example, Excel (for Microsoft Excel).
- The pipe character (|, or character code 124) separates the application from the topic.
- *topic* is the fundamental data grouping used in the source application for example, a worksheet in Microsoft Excel.

In addition, for a destination control only, you must set the related **LinkItem** property to specify the *item* element for the link. A cell reference, such as R1C1, corresponds to an item in a Microsoft Excel worksheet.

Source Form To set **LinkTopic** for a source form, set *value* to an appropriate identifier for the form. A destination application uses this string as the *topic* argument when establishing a DDE link with the form. Although this string is all you need to set **LinkTopic** within Visual Basic for a source form, the destination application also needs to specify:

- The *application* element that the destination application uses, which is either the Visual Basic project filename without the .vbp extension (if you're running your application in the Visual Basic development environment) or the Visual Basic application filename without the .exe extension (if you're running your application as a stand-alone executable file). The **EXENAME** property of the **App** object provides this string in your Visual Basic code unless the filename was changed by the user. (**EXENAME** always returns the actual filename of the application on disk; DDE always uses the original name that was specified in the Project Properties dialog box.)
- The *item* element that the destination application uses, which corresponds to the **Name** property setting for the **Label**, **PictureBox**, or **TextBox** control on the source form.

The following syntax is an example of a valid reference from Microsoft Excel to a Visual Basic application acting as a source:

`=VizBasicApplication|FormN!TextBox1`

You could enter this reference for a destination cell in the Microsoft Excel formula bar.

To activate the data link set with **LinkTopic**, set the **LinkMode** property to the appropriate nonzero value to specify the type of link you want. As a general rule, set **LinkMode** after you set **LinkTopic**. For a destination control, changing **LinkTopic** breaks an existing link and terminates the DDE conversation. For a source form, changing **LinkTopic** breaks all destination links that are using that topic. For these reasons, always set the **LinkMode** property to 0 before changing **LinkTopic**. After changing **LinkTopic** for a destination control, you must set **LinkMode** to 1 (Automatic), 2 (Manual), or 3 (Notify) to establish a conversation with the new topic.

Note Setting a permanent data link at design time with the Paste Link command on the Edit menu also sets the **LinkMode**, **LinkTopic**, and **LinkItem** properties. This creates a link that is saved with the form. Each time the form is loaded, Visual Basic attempts to reestablish the conversation.

Visual Basic Reference

LinkItem, LinkMode, LinkTopic Properties Example

In the example, each mouse click causes a cell in a Microsoft Excel worksheet to update the contents of a Visual Basic **TextBox** control. To try this example, start Microsoft Excel, open a new worksheet named Sheet1, and put some data in the first column. In Visual Basic, create a form with a **TextBox** control. Paste the code into the Declarations section, and then press F5 to run the program.

```
Private Sub Form_Click ()
    Dim CurRow As String
    Static Row    ' Worksheet row number.
    Row = Row + 1  ' Increment Row.
    If Row = 1 Then  ' First time only.
        ' Make sure the link isn't active.
        Text1.LinkMode = 0
        ' Set the application name and topic name.
        Text1.LinkTopic = "Excel|Sheet1"
        Text1.LinkItem = "R1C1"    ' Set LinkItem.
        Text1.LinkMode = 1    ' Set LinkMode to Automatic.
    Else
        ' Update the row in the data item.
        CurRow = "R" & Row & "C1"
        Text1.LinkItem = CurRow    ' Set LinkItem.
    End If
End Sub
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

List Property

[See Also](#) [Example](#) [Applies To](#)

Returns or sets the items contained in a control's list portion. The list is a string [array](#) in which each element is a list item. Available at design time for **List****Box** and **Combo****Box** controls through the Properties window; read-only at [run time](#) for **Dir****List****Box**, **Drive****List****Box**, and **File****List****Box** controls; read/write at run time for **Combo****Box** and **List****Box** controls.

Syntax

`object.List(index) [= string]`

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>index</i>	The number of a specific item in the list.
<i>string</i>	A string expression specifying the list item.

Remarks

Use this property to access list items.

For all controls except the **Dir****List****Box**, the index of the first item is 0 and the index of the last item is **List****Count****1**.

For a **Dir****List****Box** control, the index number sequence is based on the current directories and subdirectories when the control is created at run time. The directory that is currently expanded is represented using the index1. Directories above the currently expanded directory are represented by negative indexes with higher absolute values. For example,2 is the parent directory of the directory that is currently expanded and3 is the directory above that. Directories below the directory that is currently expanded range from 0 to **List****Count****1**.

Initially, **Combo****Box** and **List****Box** controls contain an empty list. For the file-system controls, the list is based on conditions that exist when the control is created at run time:

- **Dir****List****Box** contains a list of all directories, using the range -*n* to **List****Count****1**.
- **Drive****List****Box** contains the list of drive connections in effect.
- **File****List****Box** contains the list of files in the directory that is currently expanded that match the **Pattern** property. The path isn't included.

The **List** property works in conjunction with the **ListCount** and **ListIndex** properties.

For all applicable controls except a **DirListBox**, enumerating a list from 0 to **ListCount** -1 returns all items in the list. For a **DirListBox** control, enumerating the list from n to **ListCount** -1 returns a list containing all directories and subdirectories visible from the directory that is currently expanded. In this case n is the number of directory levels above the directory that is currently expanded.

Note To specify items you want to display in a **ComboBox** or **ListBox** control, use the **AddItem** method. To remove items, use the **RemoveItem** method. To keep items in alphabetic order, set the control's **Sorted** property to **True** before adding items to the list.

Using an `Option Base = 1` statement in the Declarations section doesn't affect the enumeration of elements in Visual Basic controls. The first element is always 0.

When the List index is outside the range of actual entries in the list box, a zero-length string ("") is returned. For example, `List(-1)` returns a zero-length string for a **ComboBox** or **ListBox** control.

© 2017 Microsoft

Visual Basic Reference

List Property Example

This example loads a **ComboBox** control with a list of sandwich names and displays the first item in the list. To try this example, paste the code into the Declarations section of a form that contains a **ComboBox** control, and then press F5.

```
Private Sub Form_Load ()  
    Combo1.AddItem "Denver Sandwich"    ' Add each item to list.  
    Combo1.AddItem "Reuben Sandwich"  
    Combo1.AddItem "Turkey Sandwich"  
    Combo1.Text = Combo1.List(0)    ' Display first item.  
End Sub
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

ListCount Property

[See Also](#) [Example](#) [Applies To](#)

Returns the number of items in the list portion of a control.

Syntax

object.ListCount

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

Remarks

ListCount provides specific information for each control:

- **ComboBox** and **ListBox** controls the number of items in the list.
- **DirListBox** control the number of subdirectories in the current directory.
- **DriveListBox** control the number of drive connections.
- **FileListBox** control the number of files in the current directory that match the **Pattern** property setting.

If no item is selected, the **ListIndex** property value is 1. The first item in the list is **ListIndex** = 0, and **ListCount** is always one more than the largest **ListIndex** value.

© 2017 Microsoft

Visual Basic Reference

ListCount Property Example

This example loads a list of your printer fonts into a **ComboBox** control, displays the first item in the list, and prints the total number of fonts. Each click of the command button changes all items in the list to uppercase or lowercase. To try this example, paste the code into the Declarations section of a form that contains a **ComboBox** control (**Style** = 2) and a **CommandButton** control, and then press F5 and click the **CommandButton**.

```
Private Sub Form_Load ()
    Dim I    ' Declare variable.
    AutoRedraw = True    ' Set AutoRedraw.
    For I = 0 To Printer.FontCount - 1    ' Put font names in list.
        Combo1.AddItem Printer.Fonts(I)
    Next I
    Combo1.ListIndex = 0    ' Set text to first item.
    ' Print ListCount information on form.
    Print "Number of printer fonts: "; Combo1.ListCount
End Sub
Private Sub Command1_Click ()
    Static UpperCase
    Dim I    ' Declare variable.
    For I = 0 To Combo1.ListCount - 1    ' Loop through list.
        If UpperCase Then
            Combo1.List(I) = UCase(Combo1.List(I))
        Else
            Combo1.List(I) = LCase(Combo1.List(I))
        End If
    Next I
    UpperCase = Not UpperCase    ' Change case.
End Sub
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: DataCombo/DataList Controls

Visual Studio 6.0

ListField Property

[See Also](#) [Example](#) [Applies To](#)

Returns or sets the name of the field in the **Recordset** object, specified by the **RowSource** property, used to fill the **DataCombo** or **DataList** control's list portion.

Syntax

object.**ListField** [= *value*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>value</i>	A string expression that specifies the name of a field in the Recordset specified by the RowSource property.

Remarks

The **ListField** property enables you to select which field in the **Recordset** is used to fill the list portion of the control. This property is used in conjunction with the **RowSource** property that specifies which **Data** control is used to create the **Recordset** used to fill the list.

Generally, you use two **Recordset** objects with the data-aware list controls. One **Recordset** contains a read-only list of valid selections, while the other **Recordset** is updated with selections from the list. For example, the **DataList** control could be generated from a query that returned a list of valid part numbers and their descriptions. The **ListField** property would point to the description field of the **Recordset**, so that the user doesn't see the actual part numbers. The **BoundColumn** property would point to the part number field, as this is what needs to be updated in the **Recordset**.

If the field specified by the **ListField** property can't be found in the **Recordset**, a trappable error occurs.

Data Type

String

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: Windows Controls

Visual Studio 6.0

ListImages Property

[See Also](#) [Example](#) [Applies To](#)

Returns a reference to a collection of **ListImage** objects in an **ImageList** control.

Syntax

object.ListImages

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

Remarks

You can manipulate **ListImage** objects using standard collection methods (for example, the **Add** and **Clear** methods). Each member of the collection can be accessed by its index or unique key. These are stored in the **Index** and **Key** properties, respectively, when **ListImage** is added to a collection.

© 2017 Microsoft

Visual Basic: Windows Controls

ListImages Property Example

This example adds three **ListImage** objects to a **ListImages** collection and uses them in a **ListView** control. The code refers to the **ListImage** objects using both their **Key** and **Item** properties. To try the example, place **ImageList** and **ListView** controls on a form and paste the code into the form's Declarations section. Run the example.

Note The graphics files in the code below can be found on Disk 1 of the Visual Basic or Visual Studio CDs, in the Common\Graphics directory. Change the path in the code, or copy the graphics files to your hard disk before running the code.

```
Private Sub Form_Load()  
    Dim imgX As ListImage  
    ' Add images to ListImages collection.  
    Set imgX = ImageList1.  
    ListImages.Add(,"rocket",LoadPicture("icons\industry\rocket.ico"))  
    Set imgX = ImageList1.  
    ListImages.Add(,"jet",LoadPicture("icons\industry\plane.ico"))  
    Set imgX = ImageList1.  
    ListImages.Add(,"car",LoadPicture("icons\industry\cars.ico"))  
  
    ListView1.Icons = ImageList1 ' Set Icons property.  
  
    ' Add Item objects to the ListView control.  
    Dim itmX as ListItem  
    Set itmX = ListView1.ListItems.Add()  
    ' Reference by index.  
    itmX.Icon = 1  
    itmX.Text = "Rocket"    ' Set Text string.  
    Set itmX = ListView1.ListItems.Add()  
    ' Reference by key ("jet").  
    itmX.Icon = "jet"  
    itmX.Text = "Jet"      ' Set Text string.  
    Set itmX = ListView1.ListItems.Add()  
    itmX.Icon = "car"  
    itmX.Text = "Car"      ' Set Text string.  
End Sub
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

ListIndex Property

[See Also](#) [Example](#) [Applies To](#)

Returns or sets the index of the currently selected item in the control. Not available at design time.

Syntax

`object.ListIndex` [= *index*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>index</i>	A numeric expression specifying the index of the current item, as described in Settings.

Settings

The settings for *index* are:

Setting	Description
1	(Default for ComboBox , DirListBox , and DriveListBox controls) Indicates no item is currently selected. For a ComboBox control, indicates the user has entered new text into the text box portion. For a DirListBox control, indicates the index of the current path. For a DriveListBox control, indicates the index of the current drive when the control is created at run time .
<i>n</i>	(Default for FileListBox and ListBox controls) A number indicating the index of the currently selected item.

Remarks

The expression `List(List1.ListIndex)` returns the string for the currently selected item.

The first item in the list is **ListIndex** = 0, and **ListCount** is always one more than the largest **ListIndex** value.

For a control in which users can make multiple selections, this property's behavior depends on the number of items selected. If only one item is selected, **ListIndex** returns the index of that item. In a multiple selection, **ListIndex** returns the index of the item contained within the focus rectangle, whether or not that item is actually selected.

Visual Basic Reference

ListIndex Property Example

This example displays the names of three players in a **ListBox** control and the corresponding salary of the selected player in a **Label** control. To try this example, paste the code into the Declarations section of a form that contains a **ComboBox** control and a **Label** control, and then press F5 and choose a name from the **ComboBox**.

```
Dim Player(0 To 2)    ' Dimension two arrays.
Dim Salary(0 To 2)
Private Sub Form_Load ()
    Dim I    ' Declare variable.
    AutoSize = True
    Player(0) = "Miggey McMoo"    ' Enter data into arrays.
    Player(1) = "Alf Hinshaw"
    Player(2) = "Woofers Dean"
    Salary(0) = "$234,500"
    Salary(1) = "$158,900"
    Salary(2) = "$1,030,500"
    For I = 0 To 2    ' Add names to list.
        Combo1.AddItem Player(I)
    Next I
    Combo1.ListIndex = 0    ' Display first item in list.
End Sub

Private Sub Combo1_Click ()
    ' Display corresponding salary for name.
    Label1.Caption = Salary(Combo1.ListIndex)
End Sub
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: Windows Controls

Visual Studio 6.0

ListItems Property (ListView Control)

[See Also](#) [Example](#) [Applies To](#)

Returns a reference to a collection of **ListItem** objects in a **ListView** control.

Syntax

object.**ListItems**

The *object* placeholder represents an object expression that evaluates to a **ListView** control.

Remarks

ListItem objects can be manipulated using the standard collection methods. Each **ListItem** in the collection can be accessed by its unique key, which you create and store in the **Key** property.

You can also retrieve **ListItem** objects by their display position using the **Index** property.

© 2017 Microsoft

Visual Basic: Windows Controls

Add Method (ListItems, ColumnHeaders), ListItems Property, SubItems Property Example

The following example uses the Biblio.mdb database as a source to populate a **Listview** control with **Listitem** objects. To try this example, place a **Listview** control on a form and paste the code into the Declarations section. You must also be sure that the Biblio.mdb has been installed on your machine. In the code below, check the path in the **OpenDatabase** function and change it to reflect the actual path to Biblio.mdb on your machine.

Note The example will not run unless you add a reference to the Microsoft DAO 3.51 Object Library. To do this, on the **Project** menu click **References**. Search for Microsoft DAO 3.51 Object Library and click the checkbox to select it.

```
Private Sub Form_Load()
    ' Add ColumnHeaders. The width of the columns is
    ' the width of the control divided by the number of
    ' ColumnHeader objects.
    ListView1.ColumnHeaders. _
    Add , , "Author", ListView1.Width / 3
    ListView1.ColumnHeaders. _
    Add , , "Author ID", ListView1.Width / 3, _
    lvwColumnCenter
    ListView1.ColumnHeaders. _
    Add , , "Birthdate", ListView1.Width / 3
    ' Set View property to Report.
    ListView1.View = lvwReport

    ' Declare object variables for the
    ' Data Access objects.
    Dim myDb As Database, myRs As Recordset
    ' Set the Database to the BIBLIO.MDB database.
    ' IMPORTANT: the Biblio.mdb must be on your
    ' machine, and you must set the correct path to
    ' the file in the OpenDatabase function below.
    Set myDb = DBEngine.Workspaces(0) _
    .OpenDatabase("c:\Program Files\VB\BIBLIO.MDB")
    ' Set the recordset to the "Authors" table.
    Set myRs = _
    myDb.OpenRecordset("Authors", dbOpenDynaset)

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

    ' While the record is not the last record,
    ' add a ListItem object. Use the author field for
    ' the ListItem object's text. Use the AuthorID
    ' field for the ListItem object's SubItem(1).
    ' Use the "Year of Birth" field for the ListItem
    ' object's SubItem(2).

    While Not myRs.EOF
```

```
Set itmX = ListView1.ListItems._  
Add(, , CStr(myRs!Author))    ' Author.  
  
' If the AuthorID field is not null, then set  
' SubItem 1 to it.  
If Not IsNull(myRs!Au_id) Then  
    itmX.SubItems(1) = CStr(myRs!Au_id)  
End If  
  
' If the birth field is not Null, set  
' SubItem 2 to it.  
If Not IsNull(myRs![Year Born]) Then  
    itmX.SubItems(2) = myRs![Year Born]  
End If  
myRs.MoveNext    ' Move to next record.  
Wend  
End Sub
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: Windows Controls

Visual Studio 6.0

ListSubItems Property

See Also Example [Applies To](#)

Returns a reference to the **ListSubItems** collection of **ListSubItem** objects.

Syntax

object.**ListSubItems**

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

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

LocaleID Property

See Also Example [Applies To](#)

Returns a long value that contains the Locale identification (language and country) of the user.

Syntax

object.**LocaleID**

The **LocaleID** property syntax has this part:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.

Remarks

The **LocaleID** ambient property contains identification of the language and country of the current user. Using this identification, the control can modify its behavior and appearance to fit the language and country. This could be as simple as having error notifications in the language of the user, to more complex modifications of property, method, and event names in the language of the user.

If the container does not implement this ambient property, the default value will be the current System LocaleID.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: Winsock Control

Visual Studio 6.0

LocalHostName Property

See Also Example [Applies To](#)

Returns the local machine name. Read-only and unavailable at design time.

Syntax


object.**LocalHostName**

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

Return Value

String

© 2017 Microsoft



This documentation is archived and is not being maintained.

Visual Basic: Winsock Control

Visual Studio 6.0

LocalIP Property

See Also Example [Applies To](#)

Returns the IP address of the local machine in the IP address dotted string format (xxx.xxx.xxx.xxx). Read-only and unavailable at design time.

Syntax

object.**LocalIP**

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

Data Type

String

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: Winsock Control

Visual Studio 6.0

LocalPort Property

See Also Example [Applies To](#)

Returns or sets the local port to use. Read/Write and available at design time.

- For the client, this designates the local port to send data from. Specify port 0 if the application does not need a specific port. In this case, the control will select a random port. After a connection is established, this is the local port used for the TCP connection.
- For the server, this is the local port to listen on. If port 0 is specified, a random port is used. After invoking the **Listen** method, the property contains the actual port that has been selected.

Syntax

object.**LocalPort** = *long*

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

Data Type

Long

Remarks

Port 0 is often used to establish connections between computers dynamically. For example, a client that wishes to be "called back" by a server can use port 0 to procure a new (random) port number, which can then be given to the remote computer for this purpose.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

Location Property

[See Also](#) [Example](#) [Applies To](#)

Returns a reference to a **Location** object that describes the position of textual chart elements.

Syntax

object.**Location**

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

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

LocationRect Property

See Also [Example](#) [Applies To](#)

Returns a reference to a **Rect** object that specifies the location of the chart plot using x and y coordinates.

Syntax

object.**LocationRect**

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

Remarks

The values of this property are used to position the plot if **AutoLayout** is **False**.

If this property is set, then the **AutoLayout** property is automatically set to **False**.

© 2017 Microsoft

LocationRect Property, Rect Object Example

The example increases the size of the chart plot using the **LocationRect** property and the x and y properties of the **Rect** object.

```
Private Sub Command1_Click()  
    ' Increase the size of the chart plot.  
    MSChart1.Plot.AutoLayout = False  
    With MSChart1.Plot.LocationRect  
        .Min.x = .Min.x * 1.2  
        .Min.y = .Min.y * 1.2  
        .Max.x = .Max.x * 1.2  
        .Max.y = .Max.y * 1.2  
    End With  
End Sub
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

LocationType Property

[See Also](#) [Example](#) [Applies To](#)

Returns or sets the standard position used to display a chart element.

Syntax

object.**LocationType** [=*type*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>type</i>	Integer. For the DataPointLabel object, A VtChLabelLocationType constant identifying label position. For a Location object, A VtChLocationType constant describing the location of text.

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

Locked Property

[See Also](#) [Example](#) [Applies To](#)

Returns or sets a value indicating whether a control can be edited.

Syntax

object.**Locked** [= *boolean*]

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

Part	Description
<i>Object</i>	An object expression that evaluates to an object in the Applies To list.
<i>Boolean</i>	A Boolean expression that specifies whether the control can be edited, as described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	TextBox control you can scroll and highlight the text in the control, but you can't edit it. The program can still modify the text by changing the Text property.
	Column object you can't edit the values in the column.
	ComboBox object you can't type in the textbox.
False	TextBox control you can edit the text in the control.
	Column object you can edit the values in the column.
	ComboBox object you can type in the textbox and drop down its list.

Remarks

For the **Column** object, the default setting of **Locked** is the value of the **DataUpdatable** property for the underlying field; however, if **Column** is unbound or the data source doesn't support **DataUpdatable**, the default is **True**. If **DataUpdatable** in the underlying field is **False**, you do not create an error by setting this property to **True**. However, an error will occur when the control attempts to write the changed data to the database.

For the **ComboBox** control, when **Locked** is set to **True**, the user cannot change any data, but can highlight data in the text box and copy it. This property does not affect programmatic access to the **ComboBox**.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: DataCombo/DataList Controls

Visual Studio 6.0

Locked Property (DataCombo, DataList Controls)

See Also Example [Applies To](#)

Returns or sets a value indicating whether any data in the object can be modified.

Syntax

object.**Locked** [= *boolean*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>boolean</i>	Optional. A Boolean expression that indicates whether data can change, as described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
False	(Default) The data can be changed.
True	The data cannot be changed.

This documentation is archived and is not being maintained.

Visual Basic: RichTextBox Control

Visual Studio 6.0

Locked Property (RichTextBox Control)

[See Also](#) [Example](#) [Applies To](#)

Returns or sets a value indicating whether the contents in a **RichTextBox** control can be edited.

Syntax

object.**Locked** [= *boolean*]

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

Part	Description
<i>object</i>	An object expression that evaluates to a RichTextBox control.
<i>boolean</i>	A Boolean expression specifying whether the contents of the control can be edited, as described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	You can scroll and highlight the text in the control, but you can't edit it. The program can still modify the text by changing the Text property.
False	(Default) You can edit the text in the control.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: RDO Data Control

Visual Studio 6.0

LockEdits Property (Remote Data)

[See Also](#) [Example](#) [Applies To](#)

Returns a [Boolean](#) value indicating the type of locking that is in effect.

Syntax

object.**LockEdits**

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

Return Values

The return values for **LockEdits** are:

Setting	Description
True	Pessimistic locking is in effect.
False	(Default) Optimistic locking is in effect.

Remarks

If a [page](#) is locked and the [data source](#) uses page locking, no other user can edit [rows](#) on the same page. If row-level locking is used, the row being edited and all other rows in the rowset are locked. The rowset is defined as the number of rows specified by the **RowsetSize** property. If **LockEdits** is **True** and another user already has the page locked, an error occurs when you use the **OpenResultset** method. Generally, other users can read data from locked pages.

If **LockEdits** is **False** (the default) and you later use **Update** while the page is locked by another user, an error occurs. To see the changes made to your row by another user (and lose your changes), set the **Bookmark** property of your **rdoResultset** object to itself.

Note Data page size is determined by the [data source](#). Microsoft SQL Server uses 2K data pages.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

LockType Property (DEDesigner Extensibility)

See Also Example [Applies To](#)

Sets the type of lock that the provider uses to open the **DECommand** object, and returns the type of lock used by an open **DECommand** object.

Syntax

object.**LockType** [=*value*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an item in the Applies To list.
<i>value</i>	A constant or value that specifies the type of lock to use.

Remarks

This property corresponds to the ADO Recordset LockType property.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: RDO Data Control

Visual Studio 6.0

LockType Property (Remote Data)

[See Also](#) [Example](#) [Applies To](#)

Returns or sets a [Long](#) integer value indicating the type of concurrency handling.

Syntax

object.**LockType** [= *value*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>value</i>	A constant or Long value as described in Settings.

Settings

The settings for *value* are:

Constant	Value	Description
rdConcurReadOnly	1	(Default) Cursor is read-only. No updates are allowed.
rdConcurLock	2	Pessimistic concurrency.
rdConcurRowVer	3	Optimistic concurrency based on row ID.
rdConcurValues	4	Optimistic concurrency based on row values.
rdConcurBatch	5	Optimistic concurrency using batch mode updates. Status values returned for each row successfully updated.

Remarks

In order to maintain adequate control over the data being updated, RDO provides a number of concurrency options that control how other users are granted, or refused access to the data being updated. In many cases, when you lock a particular row using one of the **LockType** settings, the remote engine might also lock the entire page containing the row. If too many

pages are locked, the remote engine might also escalate the page lock to a table lock to improve overall system performance.

Not all lock types are supported on all data sources. For example, for SQL Server and Oracle servers using the **rdUseODBC** cursor library, **static-type** **rdResultSet** objects can only support **rdConcurValues** or **rdConcurReadOnly**.

If the concurrency option is not supported by the **data source**, the driver substitutes a different concurrency option at execution time if one is available. If the driver cannot substitute a suitable alternative concurrency option, a trappable error is fired (SQLState Code 01S02 "Option Value Changed"). For **rdConcurValues**, the driver substitutes **rdConcurRowVer** and vice versa. For **rdConcurLock**, the driver substitutes, in order: **rdConcurRowVer** or **rdConcurValues**.

Choosing a Concurrency Option

Note RDO concurrency does not function as it does with Data Access Objects (DAO). Be sure to review the following sections to determine the best type of concurrency control for your application.

- **Read-Only Concurrency:** This option does not impose any exclusive locks on the rows fetched. In most cases, however, you must be granted a share lock to gain access to the rows. In other words, other users cannot have exclusive locks (read-write or intend to write locks) on the pages being accessed. Choosing this option makes the cursor read-only. This does not preclude use of action queries to update the data independent of the cursor. This is the default **LockType**.
- **Pessimistic Concurrency:** This option requests an *immediate* exclusive lock on the cursor rows which implements the lowest level of locking sufficient to ensure the row can be updated. Unlike DAO, which defers locking until the **Edit** method is used, RDO locks the first **RowsetSize** rows of the result set when the cursor is first opened with the **OpenResultSet** method. That is, if your **RowsetSize** is 100 rows, the remote engine is instructed to lock each page that contains one of these selected rows. This means up to 100 pages can be locked which can lock hundreds of rows. As the current row pointer is moved through the result set, additional pages are locked, and those no longer referenced are released. This technique assures your application that no other application is granted exclusive (read-write) access to any rows being processed by the cursor.
- **Optimistic Concurrency:** This type of concurrency management does not lock any rows or pages it simply compares the row being posted to the database with the row as it currently exists on the server. Depending on the type of optimistic concurrency chosen, RDO and the ODBC layers compare either the row ID, the row data values, TimeStamp columns or combinations of these options with existing data to determine if a row has changed since last fetched. If no changes have taken place since the last fetch, the update is made. Otherwise, your application triggers a trappable error.

The **LockType** property supports three types of optimistic concurrency as described below. When using the Optimistic Batch Concurrency option (**rdConcurBatch**), you should also set the **UpdateCriteria** property to choose an appropriate update concurrency option.

- **Optimistic Concurrency Row Version:** By comparing the row identifier (usually a TimeStamp column) , RDO can determine if the row has changed since last fetched. If it has, a trappable error results.
- **Optimistic Concurrency Row Values:** By comparing row values on a column-by-column basis, RDO can determine if the row has changed since last fetched. If it has, a trappable error results.
- **Optimistic Batch:** This type of concurrency uses the **UpdateCriteria** property to determine how to test if rows have changed when using the **UpdateBatch** method.

This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

LogBase Property

See Also Example [Applies To](#)

Returns or sets the logarithm base used to plot chart values on a logarithmic axis.

Syntax

object.**LogBase** [= *base*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>base</i>	Integer. An integer that identifies the logarithm base. The default base is 10. The valid range is 2 to 100.

Remarks

This property has an effect only when the **Type** property is set to **VtChScaleTypeLogarithmic**.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: RDO Data Control

Visual Studio 6.0

LoginTimeout Property (Remote Data)

[See Also](#) [Example](#) [Applies To](#)

Returns or sets a value that specifies the number of seconds the [ODBC driver manager](#) waits before a timeout error occurs when a [connection](#) is opened.

Syntax

object.**LoginTimeout** [= *value*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>value</i>	A Long integer representing the number of seconds the driver manager waits before timing out and returning an error.

Remarks

If *value* is 0, no timeout occurs and an error does not occur if a connection cannot be established. If you are not using asynchronous connections, this might cause your application to block indefinitely.

When you're attempting to connect to an ODBC database, such as SQL Server, there may be delays due to network traffic or heavy use of the [ODBC data source](#). Rather than waiting indefinitely, you can specify how long to wait before the ODBC driver manager produces an error.

The default timeout value is either 15 seconds or a value set by the **rdoDefaultLoginTimeout** property. When used with an **rdoEnvironment** object, the **LoginTimeout** property specifies a global value for all login operations associated with the **rdoEnvironment**. The **LoginTimeout** setting of on an **rdoConnection** object overrides the default value.

If the specified timeout exceeds the maximum timeout in the data source, or is smaller than the minimum timeout, the driver substitutes that value and the following error is logged in the **rdoErrors** collection: SQLState 01S02 "Option value changed."

Typically, a connection to a remote server on a Local Area Network (LAN) takes under eight seconds to complete. Remote Access Service (RAS) or Internet connections can take far longer depending on Wide Area Network bandwidth, load and other factors.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: RDO Data Control

Visual Studio 6.0

LogMessages Property (Remote Data)

[See Also](#) [Example](#) [Applies To](#)

Enables ODBC trace logging and returns or sets a value indicating the path of the [ODBC](#) trace file created by the [ODBC driver manager](#) to record all ODBC operations.

Syntax

object.**LogMessages** [= *value*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>value</i>	A String expression as described in Settings.

Settings

Value contains the path of an ASCII file used to log ODBC operations. If the **LogMessages** property is an empty string, no logging takes place.

Remarks

When the **LogMessages** property is **True**, all ODBC commands are sent to an ASCII log file that can be used to debug or tune queries or other operations.

On Windows NT or Windows 95 (or later), tracing should *only* be used for a single application or each application should specify a different trace file. Otherwise, two or more applications might attempt to open the same trace file at the same time, causing an error.

Note ODBC performance is adversely affected when the log is enabled.

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

LogMode Property

[See Also](#) [Example](#) [Applies To](#)

Returns a value which determines how logging (through the **LogEvent** method) will be carried out. Read-only at run time.

Syntax

object.**LogMode** = *mode*

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>mode</i>	Long. Determines the method of logging, as shown in Settings below.

Settings

The settings for *mode* are:

Constant	Value	Description
vbLogAuto	0	If running on Windows 95 or later, this option logs messages to the file specified in the LogPath property. If running on Windows NT, messages are logged to the Windows NT Application Event Log, with "VBRunTime" used as the application source and App.Title appearing in the description.
VbLogOff	1	Turns all logging off. Messages from UI shunts as well as from the LogEvent method are ignored and discarded.
VbLogToFile	2	Forces logging to a file. If no valid filename is present in LogPath , logging is ignored, and the property is set to vbLogOff .
VbLogToNT	3	Forces logging to the NT event log. If not running on Windows NT, or the event log is unavailable, logging is ignored and the property is set to vbLogOff .
VbLogOverwrite	0x10	Indicates that the logfile should be recreated each time the application starts. This value can be combined with other mode options using the OR operator. The default action for logging is to append to the existing file. In the case of NT event logging, this flag has no meaning.
VbLogThreadID	0x20	Indicates that the current thread ID be prepended to the message, in the form "[T:0nnn] ".

		This value can be combined with other mode options using the OR operator. The default action is to show the thread ID only when the application is multi-threaded (either explicitly marked as thread-safe, or implemented as an implicit multithreaded app, such as a local server with the instancing property set to Single-Use, multithreaded).
--	--	---

Return Type

Long

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: MAPI Controls

Visual Studio 6.0

LogonUI Property

[See Also](#) [Example](#) [Applies To](#)

Specifies whether or not a dialog box is provided for sign-on.

Syntax

```
object.LogonUI [ = value ]
```

The LogonUI property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>value</i>	A boolean expression specifying whether a logon dialog box is displayed, as described in Settings.

Settings

The settings for *value* are:

Setting	Description
True	(Default) A dialog box prompts new users for their user name and password (unless a valid messaging session already exists. See the NewSession property for more information).
False	No dialog box is displayed.

Remarks

The **False** setting is useful when you want to begin a mail session without user intervention, and you already have the account name and password for the user. If insufficient or incorrect values are provided, however, an error is generated.

Data Type

Boolean

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

LogPath Property

[See Also](#) [Example](#) [Applies To](#)

Returns the path and filename of the file used to capture output from the **LogEvent** method. Not available at design-time; read-only at run time.

Syntax

object.**LogPath** = *path*

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>path</i>	String. The path and filename of a log file.

Remarks

The **LogMode** property determines how logging will be carried out. If no **LogPath** is set, the **LogEvent** method writes to the NT LogEvent file.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

IpOleObject Property

See Also Example [Applies To](#)

Returns the address of the object.

Syntax

object.**IpOleObject**

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

Remarks

Many function calls in the ActiveX DLLs require the address of an object as an argument. Pass the value specified in the **IpOleObject** property when making API calls to the ActiveX DLLs. The value is 0 if no object is currently displayed. If a call is made to an API that makes a callback to the **OLE** container control, the result is unpredictable.

The address returned by this property is a pointer to the **IpOleObject** interface for the active object.

© 2017 Microsoft