# Visual Basic Reference

**Visual Studio 6.0**

# Quit Method (Add-Ins)

See Also Example Applies To

Attempts to exit Visual Basic.

**Syntax**

*object*.**Quit**

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

# Visual Basic for Applications Reference

**Visual Studio 6.0**

# Raise Method

See Also Example Applies To Specifics

Generates a run-time error.

**Syntax**

*object*.**Raise** ***number, source, description, helpfile, helpcontext***

The **Raise** method has the following object qualifier and named arguments:

| Argument | Description |
|---|---|
| *object* | Required. Always the **Err** object. |
| ***number*** | Required. Long integer that identifies the nature of the error. Visual Basic errors (both Visual Basic-defined and user-defined errors) are in the range 065535. The range 0512 is reserved for system errors; the range 51365535 is available for user-defined errors. When setting the **Number** property to your own error code in a class module, you add your error code number to the **vbObjectError** constant. For example, to generate the error number 513, assign **vbObjectError** + 513 to the **Number** property. |
| ***source*** | Optional. String expression naming the object or application that generated the error. When setting this property for an object, use the form *project.class*. If *source* is not specified, the programmatic ID of the current Visual Basic project is used. |
| ***description*** | Optional. String expression describing the error. If unspecified, the value in **Number** is examined. If it can be mapped to a Visual Basic run-time error code, the string that would be returned by the **Error** function is used as **Description**. If there is no Visual Basic error corresponding to **Number**, the "Application-defined or object-defined error" message is used. |
| ***helpfile*** | Optional. The fully qualified path to the Help file in which help on this error can be found. If unspecified, Visual Basic uses the fully qualified drive, path, and file name of the Visual Basic Help file. |
| ***helpcontext*** | Optional. The context ID identifying a topic within ***helpfile*** that provides help for the error. If omitted, the Visual Basic Help file context ID for the error corresponding to the **Number** property is used, if it exists. |

**Remarks**

All of the arguments are optional except ***number***. If you use **Raise** without specifying some arguments, and the property settings of the **Err** object contain values that have not been cleared, those values serve as the values for your error.

**Raise** is used for generating run-time errors and can be used instead of the **Error** statement. **Raise** is useful for generating errors when writing class modules, because the **Err** object gives richer information than is possible if you generate errors with

the **Error** statement. For example, with the **Raise** method, the source that generated the error can be specified in the **Source** property, online Help for the error can be referenced, and so on.

# Visual Basic for Applications Reference

## Raise Method Example

This example uses the **Err** object's **Raise** method to generate an error within an Automation object written in Visual Basic. It has the programmatic ID `MyProj.MyObject`.

```
Const MyContextID = 1010407   ' Define a constant for contextID.
Function TestName(CurrentName, NewName)
   If Instr(NewName, "bob") Then   ' Test the validity of NewName.
      ' Raise the exception
      Err.Raise vbObjectError + 513, "MyProj.MyObject", _
      "No ""bob"" allowed in your name", "c:\MyProj\MyHelp.Hlp", _
      MyContextID
   End If
End Function
```

**Visual Studio 6.0**

*Visual Basic: MSChart Control*

# RandomDataFill Method

See Also Example Applies To

Fills the data grid associated with a specific chart with randomly generated data.

**Syntax**

*object*.**RandomDataFill**

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

**Visual Studio 6.0**

*Visual Basic: MSChart Control*

# RandomFillColumns Method

See Also Example Applies To

Fills a number of data grid columns associated with a chart with random values.

**Syntax**

*object*.**RandomFillColumns** (*column, count*)

The **RandomFillColumns** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *column* | Integer. Identifies the first column you wish to fill. Columns are numbered from left to right beginning with 1. |
| *count* | Integer. Specifies the number of columns you want to fill with random data. |

**Visual Studio 6.0**

*Visual Basic: MSChart Control*

# RandomFillRows Method

See Also Example Applies To

Fills a number of data grid rows associated with a chart with random values.

**Syntax**

*object*.**RandomFillRows** (*row, count*)

The **RandomFillRows** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *row* | Integer. Identifies the first row you wish to fill. Rows are numbered from top to bottom beginning with 1. |
| *count* | Integer. Specifies the number of rows you want to fill with random data. |

# Visual Basic: RDO Data Control

**Visual Studio 6.0**

# rdoCreateEnvironment Method (Remote Data)

See Also Example Applies To

Creates a new **rdoEnvironment** object.

**Syntax**

Set *variable* = **rdoCreateEnvironment(***name*, *user*, *password***)**

The **rdoCreateEnvironment** method syntax has these parts:

| Part | Description |
|---|---|
| *variable* | An object expression that evaluates to an **rdoEnvironment** object. |
| *name* | A String variable that uniquely names the new **rdoEnvironment** object. See the **Name** property for details on valid **rdoEnvironment** names. |
| *user* | A **String** variable that identifies the owner of the new **rdoEnvironment** object. See the **UserName** property for more information. |
| *password* | A **String** variable that contains the password for the new **rdoEnvironment** object. The password can be up to 14 characters long and can include any characters except ASCII character 0 (null). |

**Remarks**

The **rdoEnvironment** object defines a transaction, user, and password context. When the **rdoEngine** is initialized, a default **rdoEnvironments**(0) is created automatically with the **rdoDefaultUser** and **rdoDefaultPassword** user name and password. If you need to define alternate transaction scopes that contain specific **rdoConnection** objects, or specific users, use the **rdoCreateEnvironment** method and specify the specific users for the environment. You can then open connections against this new environment.

Unlike the other methods you use to create Remote Data Objects, **rdoCreateEnvironment** requires that you provide all of its parts. If you dont provide all of the parts, the object wont be added to the collection. In addition, **rdoEnvironment** objects arent permanent and cant be saved. Once you create an **rdoEnvironment** object, you can only modify the **UserName** and **Timeout** property settings.

You dont have to append the new **rdoEnvironment** object to a collection before you can use it it is automatically appended to the **rdoEnvironments** collection.

If *name* refers to an object that is already a member of the **rdoEnvironments** collection, a trappable error occurs.

Once you use **rdoCreateEnvironment** to create a new **rdoEnvironment** object, an **rdoEnvironment** session is started, and you can refer to the **rdoEnvironment** object in your application.

To remove an **rdoEnvironment** object from the **rdoEnvironments** collection, use the **Close** method on the **rdoEnvironment** object. You cannot remove **rdoEnvironments**(0).

# Visual Basic: RDO Data Control

**Visual Studio 6.0**

# rdoRegisterDataSource Method (Remote Data)

See Also Example Applies To

Enters connection information for an ODBC data source into the Windows Registry.

**Syntax**

**rdoRegisterDataSource** *DSN*, *driver*, *silent*, *attributes*

The **rdoRegisterDataSource** method syntax has these parts:

| Part | Description |
|---|---|
| *DSN* | A string expression that is the name used in the **OpenConnection** method that refers to a block of descriptive information about the data source. For example, if the data source is an ODBC remote database, it could be the name of the server. |
| *driver* | A string expression that is the name of the ODBC driver. This isnt the name of the ODBC driver dynamic link library (DLL) file. For example, *SQL Server* is a driver name, but *SQLSRVR.DLL* is the name of a DLL file. You must have ODBC and the appropriate driver already installed. |
| *silent* | A Boolean value that is **True** if you dont want to display the ODBC driver dialog boxes that prompt for driver-specific information, or **False** if you do want to display the ODBC driver dialog boxes. If *silent* is **True**, *attributes* must contain all the necessary driver-specific information or the dialog boxes are displayed anyway. |
| *attributes* | A string expression that is a list of keywords to be added to the ODBC.INI file. The keywords are in a carriage-return-delimited string. |

**Remarks**

When you use the **OpenConnection** or **EstablishConnection** method, you can use a registered data source entry to provide connection information.

If the data source is already registered in the Windows Registry when you use the **rdoRegisterDataSource** method, the connection information is updated. If the **rdoRegisterDataSource** method fails for any reason, no changes are made to the Windows Registry, and an error occurs.

For more information about ODBC drivers such as SQL Server, see the Help file provided with the driver.

**Note** You are encouraged to use the Windows Control Panel 32-bit ODBC Data Sources dialog box to add new data sources, or to make changes to existing entries.

### Microsoft SQL Server Attributes

The following attributes are used when setting up DSN entries for Microsoft SQL Server drivers as extracted from the Drvssrvr.Hlp file. Other vendors drivers expose their own set of attributes that might or might not conform to this set. See the documentation provided with your driver for additional details.

| Keyword | Description |
|---|---|
| ADDRESS | The network address of the SQL Server database management system from which the driver retrieves data. |
| DATABASE | The name of the SQL Server database. |
| DESCRIPTION | A description of the data in the data source. |
| LANGUAGE | The national language to be used by SQL Server. |
| NETWORK | The network library connecting the platforms on which SQL Server and the SQL Server driver reside. |
| OEMTOANSI | Enables conversion of the OEM character set to the ANSI character set if the SQL Server client machine and SQL Server are using the same non-ANSI character set. Valid values are YES for on (conversion is enabled) and NO for off. |
| SERVER | The name of the network computer on which the data source resides. |
| TRANSLATIONDLL | The name of the DLL that translates data passing between an application and a data source. |
| TRANSLATIONNAME | The name of the translator that translates data passing between an application and a data source. |
| TRANSLATIONOPTION | Enables translation of data passing between an application and a data source. |
| USEPROCFORPREPARE | Disables generation of stored procedures for SQLPrepare. Valid values are NO for off (generation is disabled) and YES for on. The default value (set in the setup dialog box) is YES. |

### Setting the OEMTOANSI Option

If the SQL Server client computer and SQL Server are using the same non-ANSI character set, select this option. For example, if SQL Server uses code page 850 and this client computer uses code page 850 for the OEM code page, selecting this option will ensure that extended characters stored in the database are property converted to ANSI for use by Windows-based applications.

When this option is set to YES and the SQL Server client machine and SQL Server are using different character sets, you must specify a character set translator.

### Setting the Server Option

The Server option sets the name of the server. (local) can be entered as the server on a Microsoft Windows NT computer if the DSN is intended to reference a server on the local system. The user can then use a local copy of SQL Server (that listens on named pipes), even when running a non-networked version of SQL Server. Note that when the 16-bit SQL Server driver is using (local) without a network, the MS Loopback Adapter must be installed.

For more information about server names for different types of networks, see Microsoft SQL Server Setup.

### Setting the Address Option

The Address option sets the network pathname address of the SQL Server database management system (DBMS) from which the driver retrieves data. For Microsoft SQL Server you can usually omit this argument when sets it to (Default).

### Setting the Network Option

The Network attribute sets the name of the 32-bit SQL Server Net-Library DLL that the SQL Server driver uses to communicate with the network software. If this option is not provided, the SQL Server driver uses the client computers default Net-Library, which is specified in the Default Network box in the Net-Library tab of the SQL Server Client Configuration Utility.

If you create a data source using a Network Library and optionally a Network Address, ODBC SQL Server Setup will create a server name entry that you can see in the Advanced tab in the SQL Server Client Configuration Utility. These server name entries can also be used by DB-Library applications.

# Visual Basic: RDO Data Control

# rdoRegisterDataSource Method Example

The following example illustrates use of the **rdoRegisterDataSource** method to create a new ODBC data source entry.

```
Private Sub RegisterDataSource()
Dim en As rdoEnvironment
Dim cnTest As rdoConnection
Dim strAttribs As String
' Build keywords string.
strAttribs = "Description=" _
        & "SQL Server on server SEQUEL" _
  & Chr$(13) & "OemToAnsi=No" _
  & Chr$(13) & "SERVER=SEQUEL" _
  & Chr$(13) & "Network=DBNMPNTW" _
  & Chr$(13) & "Database=WorkDB" _
  & Chr$(13) & "Address=\\SEQUEL\PIPE\SQL\QUERY"

' Create new registered DSN.
rdoEngine.rdoRegisterDataSource "Example", _
         "SQL Server", True, strAttribs
' Open the database.
Set en = rdoEngine.rdoEnvironments(0)
Set cnTest = en.OpenConnection( _
  dsname:="Example", _
  Prompt:=rdDriverNoPrompt, _
  Connect:="UID=;PWD=;")


End Sub
```

# Visual Basic for Applications Reference

**Visual Studio 6.0**

# Read Method

See Also Example Applies To Specifics

**Description**

Reads a specified number of characters from a **TextStream** file and returns the resulting string.

**Syntax**

*object*.**Read(***characters***)**

The **Read** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | Required. Always the name of a **TextStream**object. |
| *characters* | Required. Number of characters you want to read from the file. |

# Visual Basic for Applications Reference

**Visual Studio 6.0**

# ReadAll Method

See Also Example Applies To Specifics

**Description**

Reads an entire **TextStream** file and returns the resulting string.

**Syntax**

*object*.**ReadAll**

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

**Remarks**

For large files, using the **ReadAll** method wastes memory resources. Other techniques should be used to input a file, such as reading a file line by line.

# Visual Basic Reference

**Visual Studio 6.0**

# ReadFromFile Method

See Also Example Applies To

Loads an object from a data file created using the **SaveToFile** method. Doesn't support named arguments.

**Syntax**

*object*.**ReadFromFile** *filenumber*

The **ReadFromFile** method syntax has these parts:

| Part | Description |
|---|---|
| *Object* | An object expression that evaluates to an object in the Applies To list. |
| *Filenumber* | Required. A numeric expression specifying the file number used when loading an object. This number must correspond to an open, binary file. |

**Remarks**

You can save an object to a data file using the **SaveToFile** or **SaveToOle1File** methods.

# Visual Basic for Applications Reference

**Visual Studio 6.0**

# ReadLine Method

See Also Example Applies To Specifics

**Description**

Reads an entire line (up to, but not including, the newline character) from a **TextStream** file and returns the resulting string.

**Syntax**

*object*.**ReadLine**

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

# Visual Basic Extensibility Reference

**Visual Studio 6.0**

# ReadProperty Method

See Also Example Applies To

Returns a string from the specified user-defined section and key in the project's .Vbp or component file.

- **VBComponent** object:
- **VBProject** object:

**Syntax**

*object*.**ReadProperty** (*key* **As String**) **As String**

*object*.**ReadProperty** (*section* **As String**, *key* **As String**) **As String**

The **ReadProperty** function syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *section* | A string expressioncontaining the name of the section where the key is found. |
| *key* | A string expressioncontaining the name of the key to return. |

**Remarks**

If the *section* or *key* area in the file is empty or doesn't exist, you'll get run-time error 5: "`Illegal function call`."

# Visual Basic Reference

**Visual Studio 6.0**

# ReadProperty Method

See Also Example Applies To

Returns a saved value from a **PropertyBag** class object.

**Syntax**

*object*.**ReadProperty(***DataName*[, *DefaultValue*]**)**

The **ReadProperty** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *DataName* | A string expression that represents a data value in the property bag. |
| *DefaultValue* | The value to be returned if no value is present in the property bag. |

**Remarks**

The **ReadProperty** method will return the value of the saved data that is represented by the string expression *DataName*, or *DefaultValue* if there is no saved value. *DataName* should match the string expression that was used to store the saved data value in the property bag.

**Note** Specifying a default value reduces the size of the file belonging to the container of the control. A line for the property is written to the file only if the value to be written is different from the default. Wherever possible, you should specify default values for the properties of the control when initializing, saving, and retrieving property values.

# Visual Basic: DataGrid Control

**Visual Studio 6.0**

# Rebind Method

See Also Example Applies To

Re-creates the **DataGrid** control properties and columns. Doesn't support named arguments.

**Syntax**

*object*.**Rebind**

**Remarks**

The **Rebind** method causes the **DataGrid** control to perform the same operations that occur when you set the **DataSource** property. The **DataGrid** control resets the columns, headings and other properties based on the current **Data** control properties.

If you have not modified the grid columns at design time, then executing the **ReBind** method will reset the columns, headings, and other properties based on the current data source.

However, if you have altered the columns in any way at design time (even if you leave the **DataField** properties blank), then the grid will assume that you wish to maintain the modified grid layout and will not automatically reset the columns.

For an unbound grid (one with its **DataMode** property set to 1), this method is similar to the **Refresh** method except that the grid attempts to restore the current and topmost rows.

**Note** To force the grid to reset the column bindings even if the columns were modified at design time, invoke the **ClearFields** method immediately before **ReBind**. Conversely, to cancel the grid's automatic layout response and force the grid to use the current column/field layout, invoke the **HoldFields** method immediately before **ReBind**.

# Visual Basic: DataGrid Control

# Rebind Method Example

This example checks a global variable to see if the user changed the table layout and reconfigures using the original table information.

```
Sub CheckForRebind_Click ()
   If UserChangedLayout Then
      DataGrid1.Rebind
   End If
End Sub
```

# Visual Basic: DataCombo/DataList Controls

**Visual Studio 6.0**

# ReFill Method

See Also Example Applies To

Re-creates the list of a **DataList** or **DBCombo** control and forces a repaint.

**Syntax**

*object*.**ReFill**

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

**Remarks**

The **ReFill** method is different than the standard **Refresh** method, which just forces a Repaint event.

# Visual Basic Reference

**Visual Studio 6.0**

# Refresh Method (ActiveX Controls)

See Also Example Applies To

Forces a complete repaint of a form or control.

**Syntax**

*object*.**Refresh**

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

**Remarks**

Use the **Refresh** method when you want to:

- Completely display one form while another form loads.
- Update the contents of a file-system list box, such as a **FileListBox** control.
- Update the data structures of a **Data** control.

**Refresh** can't be used on MDI forms_, but can be used on MDI child forms. You can't use **Refresh** on **Menu** or **Timer** controls.

Generally, painting a form or control is handled automatically while no events are occurring. However, there may be situations where you want the form or control updated immediately. For example, if you use a file list box, a directory list box, or a drive list box to show the current status of the directory structure, you can use **Refresh** to update the list whenever a change is made to the directory structure.

You can use the **Refresh** method on a **Data** control to open or reopen the database (if the **DatabaseName**, **ReadOnly**, **Exclusive**, or **Connect** property settings have changed) and rebuild the dynaset in the control's **Recordset** property.

# Visual Basic: RDO Data Control

**Visual Studio 6.0**

# Refresh Method (Remote Data)

See Also Example Applies To

Closes and rebuilds the **rdoResultset** object created by a RemoteData control or refreshes the members of the collections in the Applies To list.

**Syntax**

*object*.**Refresh**

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

**Remarks**

You can use the **Refresh** method on a **RemoteData** control to close and reopen the **rdoResultset** if the properties that describe the result set have changed. When you use the **Refresh** method, the properties and current row position is reset to the state set when the query was first run.

Once the **Refresh** method has been executed against the **RemoteData** control, all stored **rdoResultset** bookmarks are invalid.

If both the **BOF** and **EOF** property settings of the **rdoResultset** object are **True**, or the **RowCount** property is set to 0 after you use the **Refresh** method, the query didnt return any rows and the new **rdoResultset** contains no data.

Use the **Refresh** method in multi-user environments in which the database schema is subject to change to retrieve current table definitions. Using the **Refresh** method on an **rdoTables** collection fetches table names from the base tables in the database. Using **Refresh** on a specific **rdoTable** objects **rdoColumns** collection fetches the table structures including column names and data types from the base tables.

**Note** Before you can use the name of a base table in the ***name*** argument of the **OpenResultset** method, you must first use the **Refresh** method against the **rdoTables** collection to populate it. You can also populate the **rdoTables** collection by referencing one of its members by its ordinal number. For example, referencing **rdoTables**(0) will populate the entire collection.

# Visual Basic: RDO Data Control

# Refresh Method Example

The following example illustrates use of the **Refresh** method to rebuild an **rdoResultset** on the **RemoteData** control. The example resets the SQL property with a new query built using the concatenation technique. When the **Refresh** method is executed, the query is re-executed. Since the **Connect** property is not changed for each invocation of the Search procedure, the connection is not re-established each time it is opened only on the first invocation. When the **Refresh** method is complete, the bound controls reflect data from the columns returned by the query.

```
Option Explicit
Private Sub Search_Click()
On Error GoTo eh
With MSRDC1
   .Connect = "UID=;PWD=;Database=Pubs;"
   .DataSourceName = "WorkDB"
   .SQL = "Select Au_Fname " _
      & " From Authors " _
      & " Where Au_Lname like '%" _
      & AuthorWanted & "%'"
   Debug.Print .SQL
   .Refresh

   If .Resultset.EOF Then
      MsgBox "No authors on file with that last name"
   End If
End With
Exit Sub

eh:
Dim er As rdoError
For Each er In rdoErrors
Debug.Print er
Next
Resume Next
End Sub
```

# Visual Basic Reference

**Visual Studio 6.0**

# Refresh Method

See Also Example Applies To

Forces a complete repaint of a form or control.

**Syntax**

*object*.**Refresh**

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

**Remarks**

Use the **Refresh** method when you want to:

- Completely display one form while another form loads.
- Update the contents of a file-system list box, such as a **FileListBox** control.
- Update the data structures of a **Data** control.

**Refresh** can't be used on MDI forms, but can be used on MDI child forms. You can't use **Refresh** on **Menu** or **Timer** controls.

Generally, painting a form or control is handled automatically while no events are occurring. However, there may be situations where you want the form or control updated immediately. For example, if you use a file list box, a directory list box, or a drive list box to show the current status of the directory structure, you can use **Refresh** to update the list whenever a change is made to the directory structure.

You can use the **Refresh** method on a **Data** control to open or reopen the database (if the **DatabaseName**, **ReadOnly**, **Exclusive**, or **Connect** property settings have changed) and rebuild the dynaset in the control's **Recordset** property.

# Visual Basic Reference

# Refresh Method Example

This example uses the **Refresh** method to update a **FileListBox** control as test files are created. To try this example, paste the code into the Declarations section of a form with a **FileListBox** control named File1, and then run the example and click the form.

```
Private Sub Form_Click ()
   ' Declare variables.
   Dim FilName, Msg as String, I as Integer
   File1.Pattern = "TestFile.*"   ' Set file pattern.
   For I = 1 To 8   ' Do eight times.
      FilName = "TESTFILE." & I
      ' Create empty file.
      Open FilName For Output As FreeFile
      File1.Refresh   ' Refresh file list box.
      Close   ' Close file.
   Next I
   Msg = "Choose OK to remove the created test files."
   MsgBox Msg   ' Display message.
   Kill "TESTFILE.*"   ' Remove test files.
   File1.Refresh   ' Update file list box.
End Sub
```

# Visual Basic Reference

**Visual Studio 6.0**

# ReleaseInstance Method

See Also Example Applies To

Releases a **WebClass** object.

**Syntax**

*object*.**ReleaseInstance**

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

**Remarks**

If **StateManagement** is set to **wcRetainInstance**, then the **WebClass** object run-time releases the **WebClass** object. Developers should call the **ReleaseInstance** method during the processing of an HTTP request when the WebClass has completed its processing and should be terminated. The final HTTP response typically contains some UI element, for example, a hyperlink, that allows the user to navigate to another URL outside of the **WebClass**. If there is no such element then the user would need to manually enter another URL in the browser in order to navigate elsewhere. The response should not contain URLs to other **WebItems** in the **WebClass**, for example, created using **URLFor**. If such a URL is in the response and the user navigates to it, a new instance of the WebClass will be created.

# Visual Basic Extensibility Reference

**Visual Studio 6.0**

# Reload Method

See Also Example Applies To

Reloads the specified component from disk, discarding any unsaved changes.

**Syntax**

*object*.**Reload**

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

**Remarks**

Cursor position, code window and form visibility are not affected by the **Reload** method. **Reload** doesn't change the setting that indicates whether the project was edited since the last time it was saved.

# Visual Basic Reference

**Visual Studio 6.0**

# Remove Method (ActiveX Controls)

See Also Example Applies To

Removes a specific member from a collection.

**Syntax**

*object*.**Remove** *index*

The **Remove** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *index* | An integer or string that uniquely identifies the object within the collection. Use an integer to specify the value of the **Index** property; use a string to specify the value of the **Key** property. |

**Remarks**

To remove all the members of a collection, use the **Clear** method.

# Visual Basic Reference

# Remove Method (ActiveX Controls) Example

This example adds six **Panel** objects to a **StatusBar** control, creating a total of seven **Panel** objects. When you click on the form, the code checks to see how many **Panel** objects there are. If there is only one **Panel** object, the code adds six **Panel** objects. Otherwise, it removes the first panel. To try the example, place a **StatusBar** control on a form and paste the code into the Declarations section. Run the example and click on the form to remove one **Panel** object at a time, and subsequently add **Panel** objects.

```
Private Sub Form_Load()
   Dim pnlX As Panel  ' Declare object variable for Panel objects.
   Dim i As Integer

   ' Add 6 Panel objects to the single default Panel object,
   ' making 7 Panel objects.
   For i = 1 To 6
      Set pnlX = StatusBar1.Panels.Add(, , , i)
      pnlX.AutoSize = sbrSpring
   Next i
End Sub

Private Sub Form_Click()
   ' If the Count of the collection is 1, add 6 Panel objects.
   ' Otherwise, remove the first panel from the collection.
   If StatusBar1.Panels.Count = 1 Then
      Dim sbrX As Panel
      Dim i As Integer
      For i = 1 To 6 ' Each panel has its style set by i.
         Set sbrX = StatusBar1.Panels.Add(, , , i)
         sbrX.AutoSize = sbrSpring
      Next i
   Else ' Remove the first panel.
      StatusBar1.Panels.Remove 1
   End If
End Sub
```

# Visual Basic: DataGrid Control

**Visual Studio 6.0**

# Remove Method (DataGrid)

See Also Example Applies To

Removes the specified row from the **SelBookmarks** collection, or the specified **Column** object from the **Columns** collection of a **DataGrid** control.

**Syntax**

*object*.**Remove** *index*

The **Remove** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *index* | Required. An integer in the range of 0 to the **Count** property setting of the collection 1. |

**Remarks**

For the **SelBookmarks** collection, the **Remove** method removes the row specified by the *index* argument, then decrements the **SelBookmarks.Count** property by 1. If the row removed from the **SelBookmarks** collection is visible, it will be deselected in the **DataGrid** control.

For the **Columns** collection, the **Remove** method removes the column specified by the *index* argument, then decrements the **Columns.Count** property by 1.

If you specify a row that isn't in the **SelBookmarks** collection or a **Column** object that isn't in the **Columns** collection, a trappable error occurs.

# Visual Basic Reference

**Visual Studio 6.0**

# Remove Method (DEDesigner Extensibility)

See Also Example Applies To

Removes a member from a **DEConnections** or **DECommands** collection.

**Syntax**

*object*.**Remove(***index, Boolean***)**

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an item in the Applies To list. |
| *index* | An integer or string expression that specifies the index number of the DECommand or DEConnection object to remove from the collection. Use an integer to specify the value of the **Index** property; use a string to specify the value of the **Key** property. |
| *Boolean* | Optional. A Boolean expression that specifies whether any warning messages display during the remove. If **False**, the default, no warning messages display. |

**Remarks**

For the **DEAggregates**, **DEGroupingFields**, and **DERelationConditions** collections, the *Boolean* is not used. Thus, for these three collections, the **Remove** method is the same as Visual Basic's Remove method.

When removing an object using the Data Environment Extensibility Object Model, first set the PromptDelete property to **False** to prevent the confirmation dialog box from appearing.

# Visual Basic for Applications Reference

**Visual Studio 6.0**

# Remove Method

See Also Example Applies To Specifics

**Description**

Removes a key, item pair from a **Dictionary** object.

**Syntax**

*object*.**Remove(***key***)**

The **Remove** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | Required. Always the name of a **Dictionary** object. |
| *key* | Required. *Key* associated with the key, item pair you want to remove from the **Dictionary** object. |

**Remarks**

An error occurs if the specified key, item pair does not exist.

The following code illustrates use of the **Remove** method:

```
Dim a, d, i             'Create some variables
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Athens"     'Add some keys and items
d.Add "b", "Belgrade"
d.Add "c", "Cairo"
...
a = d.Remove()          'Remove second pair
```

**Visual Studio 6.0**

*Visual Basic: MSChart Control*

# Remove Method (LightSources Collection)

See Also Example Applies To

Removes a **LightSource** from the **LightSources** collection.

**Syntax**

*object*.**Remove** (*index*)

The **Remove** method syntax has these parts:

| Part | Description |
|---|---|
| *collection* | A object expression that evaluates to an object in the Applies To list. |
| *index* | Integer. A specific light source by position in the list of light sources. |

# Visual Basic: RDO Data Control

**Visual Studio 6.0**

# Remove Method (Remote Data)

See Also Example Applies To

Removes a specific member from a Remote Data **Collection** object.

**Syntax**

*object*.**Remove** *index*

The **Remove** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *index* | An integer or string that uniquely identifies the object within the collection. Use an integer to specify the value of the **Index** property; use a string to specify the value of the **Key** property. |

**Remarks**

To remove all the members of a collection, use the **Clear** method.

# Visual Basic Extensibility Reference

**Visual Studio 6.0**

# Remove Method (VBA Add-In Object Model)

See Also Example Applies To Specifics

Removes an item from a collection.

**Syntax**

*object*.**Remove(***component***)**

The **Remove** syntax has these parts:

| Part | Description |
|---|---|
| *object* | Required. An object expression that evaluates to an object in the Applies To list. |
| *component* | Required. For the **LinkedWindows** collection, an object. For the **References** collection, a reference to a type library or a project. For the **VBComponents** collection, an enumerated constant representing a class module, a form, or a standard module. |

**Remarks**

When used on the **LinkedWindows** collection, the **Remove** method removes a window from the collection of currently linked windows. The removed window becomes a floating window that has its own linked window frame.

# Visual Basic Extensibility Reference

**Visual Studio 6.0**

# Remove Method (Visual Basic Extensibility)

See Also Example Applies To

Removes an item from a collection.

**Syntax**

*object*.**Remove (***index***)**

The **Item** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *index* | Required. A variant expression specifying the name or index in the collection of the object to be accessed. |

**Remarks**

For the LinkedWindows collection, removes a window from the collection of currently linked windows. The removed window becomes a floating window that has its own LinkedWindowFrame. This is the point at which LinkedWindowFrame windows are created.

For the VBProjects collection, removes the specified project from the collection.

For the References collection, removes the specified reference from the collection.

# Visual Basic for Applications Reference

**Visual Studio 6.0**

# Remove Method

See Also Example Applies To Specifics

Removes a member from a **Collection** object.

**Syntax**

*object*.**Remove** *index*

The **Remove** method syntax has the following object qualifier and part:

| Part | Description |
|---|---|
| *object* | Required. An object expression that evaluates to an object in the Applies To list. |
| *index* | Required. An expression that specifies the position of a member of the collection. If a numeric expression, *index* must be a number from 1 to the value of the collection's **Count** property. If a string expression, *index* must correspond to the ***key*** argument specified when the member referred to was added to the collection. |

**Remarks**

If the value provided as *index* doesnt match an existing member of the collection, an error occurs.

# Visual Basic for Applications Reference

## Remove Method Example

This example illustrates the use of the **Remove** method to remove objects from a **Collection** object, `MyClasses`. This code removes the object whose index is 1 on each iteration of the loop.

```
Dim Num, MyClasses
For Num = 1 To MyClasses.Count
   MyClasses.Remove 1   ' Remove the first object each time
         ' through the loop until there are
         ' no objects left in the collection.
Next Num
```

# Visual Basic Reference

**Visual Studio 6.0**

# RemoveAddInFromToolbar Method

See Also Example Applies To

Removes a button from the Add-In toolbar which references an add-in or Wizard.

**Syntax**

*object*.**RemoveAddInFromToolbar** (*saddinname* **As String**)

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *saddinname* | Required. A string expression specifying the name of the add-in or Wizard to remove from the Add-In toolbar (as specified by the *saddinname* parameter from the **AddToAddInToolbar** method). |

# Visual Basic Reference

# RemoveAddInFromToolbar Method Example

This example removes an existing button from the Add-In toolbar that references a fictitious add-in called MyAddIn Title:

```
Sub Main()
  dim x as Object
  Set x=CreateObject("AddInToolbar.Manager")
  x.RemoveAddInFromToolbar sAddInName:="MyAddIn Title"
End Sub
```

# Visual Basic for Applications Reference

**Visual Studio 6.0**

# RemoveAll Method

See Also Example Applies To Specifics

**Description**

The **RemoveAll** method removes all key, item pairs from a **Dictionary** object.

**Syntax**

*object*.**RemoveAll**

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

**Remarks**

The following code illustrates use of the **RemoveAll** method:

```
Dim a, d, i             'Create some variables
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Athens"     'Add some keys and items
d.Add "b", "Belgrade"
d.Add "c", "Cairo"
...
a = d.RemoveAll         'Clear the dictionary
```

# Visual Basic: MSFlexGrid/MSHFlexGrid Controls

**Visual Studio 6.0**

# RemoveItem Method (MSHFlexGrid)

See Also Example Applies To

Removes a row from an **MSHFlexGrid** at run time. This property doesnt support named arguments.

**Syntax**

*object*.**RemoveItem(***index, number***)**

The **RemoveItem** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *index* | An integer representing the row within the **MSHFlexGrid** to remove. For the first row, *index*=0. |
| *number* | A Long value that specifies the band from which to remove the row. |

**Remarks**

This method deletes the entire row specified. To clear data without removing the rows, use the **Clear** method.

If the **BandDisplay** property is set to horizontal and the **MSHFlexGrid** is bound to a hierarchical Recordset, *number* is required. If the **BandDisplay** property is set to vertical, *number* is only needed if the band is ambiguous.

When removing a row within a band that contains child records, the child records are automatically removed.

# Visual Basic: MSFlexGrid/MSHFlexGrid Controls

# AddItem, RemoveItem Methods (MSHFlexGrid) Example

This example uses the **AddItem** method to add 100 items to an **MSHFlexGrid**. To use this example, paste the code into the Declarations section of a form with an **MSHFlexGrid** named MSHFlexGrid1, press F5, and then click the form.

**Note** If you are using the **MSFlexGrid**, substitute "MSHFlexGrid1" with "MSFlexGrid1."

```
Private Sub Form_Click ()
   Dim Entry, i, Msg               ' Declare variables.
   Msg =_
   "Choose OK to add 100 items to your MSHFlexGrid."
   MsgBox Msg   ' Display message.
   MSHFlexGrid1.Cols =2         ' Two strings per row.
   For i =1 To 100   ' Count from 1 to 100.
      Entry ="Entry " & Chr(9) & I    ' Create entry.
      MSHFlexGrid1.AddItem Entry      ' Add entry.
   Next i
   Msg ="Choose OK to remove every other entry."
   MsgBox Msg                     ' Display message.
   For i =1 To 50               ' Determine how to
      MSHFlexGrid1.RemoveItem i    ' remove every other
   Next I                        ' item.
   Msg ="Choose OK to clear all items."
   MsgBox Msg                     ' Display message.
   MSHFlexGrid1.Clear               ' Clear list box.
End Sub
```

# Visual Basic Reference

**Visual Studio 6.0**

# RemoveItem Method

See Also Example Applies To

Removes an item from a **ListBox** or **ComboBox** control or a row from an **MS Flex Grid** control. Doesn't support named arguments.

**Syntax**

*object*.**RemoveItem** *index*

The **RemoveItem** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | Required. An object expression that evaluates to an object in the Applies To list. |
| *index* | Required. Integer representing the position within the object of the item or row to remove. For the first item in a **ListBox** or **ComboBox** or for the first row in an **MS Flex Grid** control, *index* = 0. |

**Remarks**

A **ListBox** or **ComboBox** that is bound to a **Data** control doesn't support the **RemoveItem** method.

# Visual Basic Reference

# RemoveItem Method Example

This example uses the **RemoveItem** method to remove entries from a list box. To try this example, paste the code into the Declarations section of a form with a **ListBox** control named List1, and then press F5 and click the form.

```
Private Sub Form_Click ()
   Dim Entry, I, Msg   ' Declare variables.
   Msg = "Choose OK to add 100 items to your list box."
   MsgBox Msg   ' Display message.
   For I = 1 To 100   ' Count from 1 to 100.
      Entry = "Entry " & I   ' Create entry.
      List1.AddItem Entry   ' Add the entry.
   Next I
   Msg = "Choose OK to remove every other entry."
   MsgBox Msg   ' Display message.
   For I = 1 To 50   ' Determine how to
      List1.RemoveItem I   ' remove every other
   Next I   ' item.
   Msg = "Choose OK to remove all items from the list box."
   MsgBox Msg   ' Display message.
   List1.Clear   ' Clear list box.
End Sub
```

# Visual Basic Reference

**Visual Studio 6.0**

# Render Method

See Also Example Applies To

Draws all or part of a source image to a destination object.

**Syntax**

*object*.**Render**(*hdc*, *xdest*, *ydest*, *destwid*, *desthgt*, *xsrc*, *ysrc*, *srcwid*, *srchgt*, *wbounds*)

The **Render** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | Required. An object expression that evaluates to an object in the Applies To list. |
| *hdc* | Required. The handle to the destination object's device context. |
| *xdest* | Required. The x-coordinate of upper left corner of the drawing region in the destination object. This coordinate is in the scale units of the destination object. |
| *ydest* | Required. The y-coordinate of upper left corner of the drawing region in the destination object. This coordinate is in the scale units of the destination object. |
| *destwid* | Required. The width of drawing region in the destination object, expressed in the scale units of the destination object. |
| *desthgt* | Required. The height of drawing region in the destination object, expressed in the scale units of the destination object. |
| *xsrc* | Required. The x-coordinate of upper left corner of the drawing region in the source object. This coordinate is in HIMETRIC units. |
| *ysrc* | Required. The y-coordinate of upper left corner of the drawing region in the source object. This coordinate is in HIMETRIC units. |
| *srcwid* | Required. The width of drawing region in the source object, expressed in HIMETRIC units. |
| *srchgt* | Required. The height of drawing region in the source object, expressed in HIMETRIC units. |
| *wbounds* | Required. The world bounds of a metafile. This argument should be passed a value of Null unless drawing to a metafile, in which case the argument is passed a user-defined type corresponding to a RECTL structure. |

### Remarks

The recommended way to paint part of a graphic into a destination is through the **PaintPicture** method.

# Visual Basic Extensibility Reference

**Visual Studio 6.0**

# ReplaceLine Method

See Also Example Applies To Specifics

Replaces an existing line of code with a specified line of code.

**Syntax**

*object*.**ReplaceLine(***line*, *code***)**

The **ReplaceLine** syntax has these parts:

| Part | Description |
|---|---|
| *object* | Required. An object expression that evaluates to an object in the Applies To list. |
| *line* | Required. A Long specifying the location of the line you want to replace. |
| *code* | Required. A String containing the code you want to insert. |

# Visual Basic Extensibility Reference

## ReplaceLine Method Example

The following example has two steps. The first **ForNext** loop uses the **InsertLines** method to insert into `CodePanes(1)` 26 ever-longer initial segments of the alphabet, starting with a. The last line inserted is the entire alphabet.

The second **ForNext** loop uses the **ReplaceLine** method to replace each even-numbered line with the last letter in the string that previously occupied that line. Odd-numbered lines are unchanged.

```
For I = 1 to 26
   Application.VBE.CodePanes(1).CodeModule.InsertLines I, Mid$("abcdefghijklmnopqrstuvwxyz", 1, I)
Next I
For I = 1 to 13
   Application.VBE.CodePanes(1).CodeModule.ReplaceLine 2*I, Mid$("abcdefghijklmnopqrstuvwxyz", 1, I)
Next I
```

# Visual Basic: MAPI Controls

**Visual Studio 6.0**

# Reply Method

See Also Example Applies To

Replies to a message.

**Syntax**

*object*.**Reply**

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

**Remarks**

This method copies the currently indexed message to the compose buffer and adds **RE:** to the beginning of the Subject line. It also sets the **MsgIndex** property to -1.

The currently indexed message originator becomes the outgoing message recipient.

# Visual Basic: MAPI Controls

**Visual Studio 6.0**

# ReplyAll Method

See Also Example Applies To

Replies to all message recipients.

**Syntax**

*object*.**ReplyAll**

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

**Remarks**

This method copies the currently indexed message to the compose buffer and adds **RE:** to the beginning of the Subject line. It also sets the **MsgIndex** property to -1.

The message is sent to the currently indexed message originator and to all **To:** and **CC:** recipients.

# Visual Basic: RDO Data Control

**Visual Studio 6.0**

# Requery Method (Remote Data)

See Also Example Applies To

Updates the data in an **rdoResultset** object by re-executing the query on which the object is based.

**Syntax**

*object*.**Requery** [*options*]

The **Requery** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | The *object* placeholder represents an object expression that evaluates to an object in the Applies To list. |
| *options* | A Variant or constant that determines how the query is run, as specified in Settings. |

**Settings**

You can use the following constant for the *options* part:

| Constant | Value | Description |
|---|---|---|
| **rdAsyncEnable** | 32 | Execute operation asynchronously. |

**Remarks**

Use this method to ensure an **rdoResultset** contains the most recent data. When you use **Requery**, all changes made to the data in the underlying table(s) by you and other users is displayed in the **rdoResultset**, and the first row in the **rdoResultset** becomes the current row.

If you use the **rdAsyncEnable** option, control returns to your application as soon as the query is begun, often before a result set is available. To test for completion of the query, use the **StillExecuting** property. The **rdoResultset** object is not valid until **StillExecuting** returns **False**. You can also use the QueryCompleted event to indicate when the query is completed.

If the **rdoParameter** objects have changed, their new values are used in the query used to generate the new **rdoResultset**.

Once the **Requery** method has been executed, all previously stored **rdoResultset** bookmarks are invalid.

If both the **BOF** and **EOF** property settings of the **rdoResultset** object are **True**, or the **RowCount** property is set to 0 after you use the **Requery** method, the query didnt return any rows and the **rdoResultset** contains no data.

You cant use the **Requery** method on **rdoResultset** objects whose **Restartable** property is set to **False**.

# Visual Basic: RDO Data Control

# Requery Method Example

The following example illustrates use of the **Requery** method to re-execute an **rdoQuery**. Note that the **rdoResultset** is created only at form load and only re-executed on each invocation of the **Requery** method.

```
Option Explicit
Dim Cn As New rdoConnection
Dim Rs As rdoResultset
Dim Col As rdoColumn
Dim Qy As rdoQuery
Dim SQL As String
Dim TimeExpected As Single
Dim Ts As Single, Tn As Single


Private Sub SpWho_Click()
Rs.Cancel
With Rs
   .Requery
   While .StillExecuting
      SpinGlobe
      DoEvents
   Wend
   ShowRS
End With

End Sub
Sub ShowRS()
With Rs
   Form1.Cls
   For Each Col In .rdoColumns
      Print Col.Name,
   Next
   Print
   Do Until .EOF
      For Each Col In .rdoColumns
         Print Col,
      Next
      Print
      .MoveNext
   Loop
End With
End Sub
Sub SpinGlobe()
' Animate a globe here to show query is in progress.
Print ".";
End Sub

Private Sub Form_Load()
With Cn
   .Connect = "UID=;PWD=;Database=WorkDB;" _
   & "Server=sequel;Driver={SQL Server}" _
   & "DSN='';"
```

```
    .LoginTimeout = 5
    .EstablishConnection rdDriverNoPrompt, True
    Set Qy = .CreateQuery("SpWho", _
    "{ call master..sp_who (?) }")
    Qy.RowsetSize = 1
    Set Rs = Qy.OpenResultset(rdOpenForwardOnly, _
    rdConcurReadOnly, rdAsyncEnable)
    Show
    ShowRS
End With
End Sub
```

**Visual Studio 6.0**

*Visual Basic: MSChart Control*

# ResetCustom Method

See Also Example Applies To

Resets any custom attributes placed on a data point to the series default.

**Syntax**

*object*.**ResetCustom**

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

**Visual Studio 6.0**

*Visual Basic: MSChart Control*

# ResetCustomLabel Method

See Also Example Applies To

Resets any custom attributes placed on a data point label in a chart to the series default.

**Syntax**

*object*.**ResetCustomLabel**

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

# Visual Basic: MAPI Controls

**Visual Studio 6.0**

# ResolveName Method

See Also Example Applies To

Resolves the name of the currently indexed recipient.

**Syntax**

*object*.**ResolveName**

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

**Remarks**

This method searches the address book for a match on the currently indexed recipient name. If no match is found, an error is returned. It does not provide additional resolution of the message originator's name or address.

The **AddressResolveUI** property determines whether to display a dialog box to resolve ambiguous names.

This method may cause the **RecipType** property to change.

# Visual Basic: Windows Controls

**Visual Studio 6.0**

# RestoreToolbar Method

See Also Example Applies To

Restores a toolbar, created with a **Toolbar** control, to its original state after being customized.

**Syntax**

*object*.**RestoreToolbar(***key* **As String**, *subkey* **As String**, *value* **As String)**

The **RestoreToolbar** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | Required. An object expression that evaluates to a **Toolbar** control. |
| *key* | Required. A string expression that specifies the key in the Windows registry where the method retrieves the **Toolbar** information. |
| *subkey* | Required. A string expression that specifies a subkey under the *key* parameter in the registry. |
| *value* | Required. A string expression that identifies the value under the *subkey* where the **Toolbar** information is stored in the registry. |

**Remarks**

**Warning** When the **RestoreToolbar** method is used, any toolbar buttons that do not contain **ImageList ListImage** object will disappear. A user can make them visible again by using the Reset button on the Customize Toolbar dialog box. You can use the **Customize** method to programmatically invoke this dialog box for the user.

To customize the **Toolbar** control at run time, use the **Customize** method in code or if the **AllowCustomize** property is **True**, the user can customize it by double clicking the control.

The state of the toolbar can be saved in the registry using the **SaveToolbar** method. The **RestoreToolbar** method restores the state of a toolbar by reading the registry.

The following code restores the **Toolbar** control's settings for the current user, assuming they have previously been saved with the **SaveToolbar** method.

```
Toolbar1.RestoreToolbar "AppName", "User1", "Toolbar1"
```

# Visual Basic Reference

**Visual Studio 6.0**

# Resync Method (Remote Data)

See Also Example Applies To

Fetches the batch conflict values for the current row.

**Syntax**

*object*.**Resync**

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

**Remarks**

The **Resync** method is valid only when using Client-Batch Cursors.

**Resync** resynchronizes the columns in the current row in the cursor library with the current data on the server (visible to your transaction). If you have not modified the row, this method changes the **Value** and **OriginalValue** properties to match what is currently on the server.

If you have modified the row, this method will only adjust the **OriginalValue** property so as not to loose your edits. This second case is useful when you want to avoid an optimistic concurrency conflict.

The last case where this is used is when you're dealing with a row that you attempted to update using **BatchUpdate**, but a conflict occurred because the concurrency check failed. In this case, this method will adjust the **BatchConflictValue** to reflect the most recent version of the column on the server.

# Visual Basic: DataGrid Control

**Visual Studio 6.0**

# RowBookmark Method

See Also Example Applies To

Returns a value containing a bookmark for a visible row in the **DataGrid** control. Doesn't support named arguments.

**Syntax**

*object*.**RowBookmark** *value*

The **RowBookmark** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *value* | Required. An integer in the range of 0 to the setting of the **DataGrid** control's **VisibleRows** property minus 1. |

**Remarks**

**RowBookmark**(0) returns the same bookmark as the **FirstRow** property of the **DataGrid** control. The current row, as determined by the **DataGrid** control's **Bookmark** property, may not be returned by this method if the current row isn't visible.

**Note** The bookmarks returned by **RowBookmark** should not be saved because their values change as soon as rows visible in the **DataGrid** control change.

# Visual Basic: DataGrid Control

# RowBookmark Method Example

This example selects all the rows that are currently visible on the grid.

```
Sub SelectAllVisible_Click ()
   Dim I
   For I = 0 To DataGrid1.VisibleRows - 1
      DataGrid1.SelBookmarks.Add DataGrid1.RowBookmark(I)
   Next I
End Sub
```

# Visual Basic: DataGrid Control

**Visual Studio 6.0**

# RowContaining Method

See Also Example Applies To

Returns a value corresponding to the row number of the specified vertical (Y) coordinate of the **DataGrid** control. Doesn't support named arguments.

### Syntax

*object*.**RowContaining** *coordinate*

The **RowContaining** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *coordinate* | Required. A single numeric expression that specifies a vertical coordinate (Y value) based on the coordinate system of the container. |

### Remarks

The **RowContaining** method returns a value that corresponds to one of the column indexes of the control specified by *object*. This value ranges from 0 to the setting of the **VisibleRows** property -1. This method is useful when working with mouse and drag events when you are trying to determine where the user clicked or dropped another control in terms of a row of the **DataGrid** control.

If *coordinate* is outside of the coordinate system of the container, a trappable error occurs.

# Visual Basic: DataGrid Control

# RowContaining, ColContaining Method Example

This example saves the value of the cell where the user began a drag method.

```
Dim SaveValue
Sub DataGrid1_MouseDown (Button As Integer, Shift As Integer, _
 X As Single, Y As Single)
   Dim RowValue, ColValue
   ' Get the value of the row and column that the mouse is over
   RowValue = DataGrid1.RowContaining(Y)
   ColValue = DataGrid1.ColContaining(X)
   ' If the values are both valid, save the text of the cell and
   ' begin dragging.
   If RowValue > 0 And RowValue < DataGrid1.VisibleRows And _
    ColValue > 0 And ColValue < DataGrid1.VisibleCols Then
      SaveValue = DataGrid1.Columns(ColValue). _
      CellValue(DataGrid1.RowBookmark(RowValue))
      DataGrid1.Drag 1
   End If
End Sub
```

# Visual Basic: DataGrid Control

**Visual Studio 6.0**

# RowTop Method

See Also Example Applies To

Returns a value containing the Y coordinate of the top of a specified row of a **DataGrid** control. Doesn't support named arguments.

**Syntax**

*object*.**RowTop** *value*

The **RowTop** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *value* | Required. An integer that specifies a row in the range of 0 to the setting of the **VisibleRows** property -1. |

**Remarks**

The **RowTop** method returns a value that corresponds to the Y coordinate of the top of the row specified by *value*. The return value is based on the **ScaleMode** property of the container.

You can use the **RowTop** method with the **RowHeight**, **Left**, and **Width** properties of the **Column** object to determine the exact location and dimension of a chosen cell in the **DataGrid** control.

# Visual Basic: DataGrid Control

# RowTop Method Example

This example begins a drag operation in the grid. Using the grid cell location and size properties, a **Label** control the size of the cell is used as the drag object.

```
Sub DataGrid1_MouseDown (Button As Integer, _
Shift As Integer, X As Single, Y As Single)
   ' Declare variables.
   Dim DY, DX, RowValue, ColValue, CellLeft, CellTop
 ColValue = DataGrid1.ColContaining(X)
 RowValue = DataGrid1.RowContaining(Y)
   ' Get the height of the cell.
 DY = DataGrid1.RowHeight
   ' Get the width of the cell.
 DX = DataGrid1.Columns(ColValue).Width
   CellLeft = DataGrid1.Columns(ColValue).Left
 CellTop = DataGrid1.RowTop(RowValue)
 Label1.Caption = DataGrid1.Columns(ColValue). _
      CellValue(DataGrid1.RowBookmark(RowValue))
 Label1.Move CellLeft, CellTop, DX, DY
 Label1.Drag ' Drag label outline.
End Sub
```