

This documentation is archived and is not being maintained.

Visual Basic: RDO Data Control

Visual Studio 6.0

BatchUpdate Method (Remote Data)

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

Performs a batched optimistic update.

Syntax

object.**BatchUpdate** (*SingleRow*, *Force*)

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>SingleRow</i>	A Boolean value that is True if the update is done only for the current row, or False if the update applies to all rows in the batch. Default is False .
<i>Force</i>	A Boolean value that is True if the row or batch of rows will overwrite existing rows in the database regardless if they cause collisions or not. Default is False .

Remarks

This method performs a batch optimistic update operation. When using batch optimistic concurrency, it is necessary to call this method to actually send the changes back to the server.

Batch updates are used whenever you open a connection using the Client Batch cursor library (**rdUseClientBatch**). In this case, each time you use the **Update** or **UpdateRow** methods, the local **rdoResultset** is updated, but the base database tables are not changed. The **BatchUpdate** method is used to update the base database table(s) with any information changed since the **rdoResultset** was last created or synchronized with the **BatchUpdate** command.

The **BatchUpdate** method updates the **BatchCollisionRows** property to include a bookmark for each row that failed to update collided with an existing row that has data more current than the **rdoResultset** object as it existed when first read. The **BatchCollisionCount** property indicates how many collisions occurred during the batch update process.

If you use the **CancelBatch** method, the changes saved to the local **rdoResultset** object are discarded. When you use the **CancelUpdate** method, only the current rows changes are rolled back to the state prior to execution of the last **Update** method.

The **SingleRow** parameter can be used in conjunction with the **Force** parameter to force the clients version of the data back into the database, even if collisions have occurred. The **SingleRow** parameter will tell RDO to only send the current row back to the server and not the entire batch, and the **Force** parameter will tell RDO to force the data in, and not use the normal optimistic concurrency detection.

Setting both the **SingleRow** and **Force** parameters to **True** overlays a single database row with the current updated **rdoResultset** row. This is useful when processing collision rows and you want to force your local version of the data to be saved regardless of the current database row setting.

Setting **SingleRow** to **False** and **Force** to **True** will cause all rows that are dirty to be forced into the database, which is useful as a shorthand way of forcing everything in (the last-one-in-wins scenario).

Setting **SingleRow** to **True** and **Force** to **False** will cause just the current row to go through the optimistic concurrency update, which is useful when you only want to update the current row, not the entire batch.

© 2018 Microsoft

 This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

BeginQueryEdit Method

See Also Example [Applies To](#)

Returns or sets the **DECommand** object which the Query designer will begin to edit.

Note The Data Environment designer cannot edit the **DECommand** object until the **EndQueryEdit** method is called.

Syntax

object.**BeginQueryEdit**

The **BeginQueryEdit** method syntax has one part:

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

Remarks

This property enables communication between Data View and the Data Environment designer.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: RDO Data Control

Visual Studio 6.0

BeginTrans, CommitTrans, RollbackTrans Methods (Remote Data)

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

The transaction methods manage [transaction](#) processing during a [session](#) represented by the *object* placeholder as follows:

- **BeginTrans** begins a new transaction.
- **CommitTrans** ends the [current transaction](#) and saves the changes.
- **RollbackTrans** ends the current transaction and restores the [databases](#) in the **rdoEnvironment** object to the state they were in when the current transaction began.

You can use the transaction methods with an **rdoConnection** object but in this case, the transaction scope only includes **rdoResultset** and **rdoQuery** objects created under the **rdoConnection**.

Syntax

object.**BeginTrans** | **CommitTrans** | **RollbackTrans**

The *object* placeholder represents an [object expression](#) that evaluates to an object in the Applies To list.

Remarks

You use the transaction methods with an **rdoEnvironment** or **rdoConnection** object when you want to treat a series of changes made to the databases in a session as one logical unit. That is, either the set of operations completes as a set, or is rolled back as a set. This way if any operation in the set fails, the entire transaction fails. Transactions also permit you to make *temporary* changes to the database changes that can be undone with the **RollbackTrans** method.

Typically, ODBC drivers work in one of two modes:

- **Auto-commit Mode:** When you have not explicitly started a transaction using the **BeginTrans** method, every operation executed is immediately committed to the database upon completion.
- **Manual-commit Mode:** When you explicitly start a transaction using the **BeginTrans** method or use the ODBC **SQLSetStmtOption** function to disable the SQL_AUTO_COMMIT mode, or send an SQL statement to begin a transaction (BEGIN TRANS), operations are part of a transaction and no changes are committed to the database until you use the **CommitTrans** method. If the connection fails before **CommitTrans** is executed, or you use the **RollbackTrans** method, the operations are undone rolled back.

Note When working with remote servers that support a Distributed Transaction Coordinator (DTC) like Microsoft SQL Server, you can initiate and control transactions that span more than one server. That is, if you invoke a procedure on the remote server that invokes a remote procedure call, the DTC service can ensure that this operation is included in the initial transaction.

Typically, you use transactions to maintain the integrity of your data when you must [update rows](#) in two or more [tables](#) and ensure that changes made are completed (committed) in all tables or none at all (rolled back). For example, if you transfer money from one account to another, you might subtract an amount from one and add the amount to another. If either update fails, the accounts no longer balance. Use the **BeginTrans** method before updating the first row, and then, if any subsequent update fails, you can use the **RollbackTrans** method to undo all of the updates. Use the **CommitTrans** method after you successfully update the last row.

Caution Within one **rdoEnvironment** object, transactions are always global to the **rdoEnvironment** and are not limited to only one database or [result set](#). If you perform operations on more than one database or result set within an **rdoEnvironment** transaction, the **RollbackTrans** method restores all operations on those databases and result sets.

Once you use **CommitTrans**, you can't undo changes made during that transaction unless the transaction is nested within another transaction that is itself rolled back. You cannot nest transactions unless you use an [action query](#) to directly execute [SQL](#) transaction management statements. If you want to have simultaneous transactions with overlapping, non-nested scopes, you can create additional **rdoEnvironment** objects to contain the concurrent transactions.

Note You can use SQL action queries that contain transaction statements. For example, with Microsoft SQL Server, you can use SQL statements like BEGIN TRANSACTION, COMMIT TRANSACTION, or ROLLBACK TRANSACTION. This technique supports nested transactions which may not be supported by the [ODBC driver](#).

If you close an **rdoEnvironment** object without saving or rolling back any pending transactions, the transactions are automatically rolled back.

No error occurs if you use the **CommitTrans** or **RollbackTrans** method without first using the **BeginTrans** method.

Some databases may not support transactions, in which case the **Transactions** property of the **rdoConnection** object or **rdoResultset** object is **False**. To make sure that the database supports transactions, check the value of the **Transactions** property of the **rdoConnection** object before using the **BeginTrans** method. If you are using an **rdoResultset** object based on more than one database, check the **Transactions** property of the **rdoResultset** object. If the **rdoConnection** or **rdoResultset** doesn't support transactions, the methods are ignored and no error occurs.

This documentation is archived and is not being maintained.

Visual Basic: Winsock Control

Visual Studio 6.0

Bind Method

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

Specifies the LocalPort and LocalIP to be used for TCP connections. Use this method if you have multiple protocol adapters.

Syntax

object.**Bind** *LocalPort*, *LocalIP*

The **Bind** method syntax has these parts

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>LocalPort</i>	The port used to make a connection.
<i>LocalIP</i>	The local Internet address used to make a connection.

Remarks

You must invoke the **Bind** method before invoking the **Listen** method.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic for Applications Reference

Visual Studio 6.0

BuildPath Method

See Also Example Applies To Specifics

Description

Appends a name to an existing path.

Syntax

object.**BuildPath**(*path*, *name*)

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

Part	Description
<i>object</i>	Required. Always the name of a FileSystemObject .
<i>path</i>	Required. Existing path to which <i>name</i> is appended. Path can be absolute or relative and need not specify an existing folder.
<i>name</i>	Required. Name being appended to the existing <i>path</i> .

Remarks

The **BuildPath** method inserts an additional path separator between the existing path and the name, only if necessary.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: RDO Data Control

Visual Studio 6.0

Cancel Method (Remote Data)

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

Cancels the processing of a [query](#) running in [asynchronous](#) mode, or cancels any pending results against the specified RDO object.

Syntax

object.**Cancel**

The *object* placeholder represents an [object expression](#) that evaluates to an object in the Applies To list.

Remarks

The **Cancel** method *requests* that the remote [data source](#) stop work on a pending asynchronous query or cancels any pending results. In some cases, it might not be possible to cancel an operation once it is started, and in other cases it might be possible to cancel the operation, but part of its steps might have already been completed.

In situations where you need to create a [result set](#), but do not want to wait until the query engine completes the operation, you can use the **rdAsyncEnable** option with the **OpenResultset** or **Execute** method. This option returns control to your application as soon as the operation is initiated, but before the first [row](#) is ready for processing. This gives you an opportunity to execute other code while the query is executed. If you need to stop this operation before it is completed, use the **Cancel** method against the object being created.

The **Cancel** method can also be used against an **rdoConnection** object when you use the **rdAsyncEnable** option to request an asynchronous connection. In this case the attempt to connect to the remote server is abandoned.

You can also use the **Cancel** method against a synchronous **rdoResultset** or **rdoQuery** object to flush remaining result set rows and release resources committed to the query and **rdoResultset**.

If you use the **Cancel** method against **rdoResultset** objects that have multiple result sets pending, *all* result sets are flushed. To simply cancel the *current* set of results and begin processing the next set, use the **MoreResults** method.

Note Using the **Cancel** method against an executing [action query](#) might have unpredictable results. If the query is performing an operation that affects a number of rows, some of the rows might be changed, while others are not. For example, if you execute an action query containing an [SQL UPDATE](#) statement and use the **Cancel** method before the operation is complete, an indeterminate number of rows are updated leaving others unchanged. If you intend to use the **Cancel** method against this type of action query, it is recommended that you use transaction methods to rollback or commit partially completed operations.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: Internet Control

Visual Studio 6.0

Cancel Method

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

Cancels the current request and closes any connections currently established.

Syntax

object.**Cancel**

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

Return Value

None

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

CancelAsyncRead Method

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

Cancels an asynchronous data request.

Syntax

object.**CancelAsyncRead** [*PropertyName*]

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>PropertyName</i>	An optional string expression specifying the name of the asynchronous data request to cancel.

Remarks

Only the asynchronous data read request specified by *PropertyName* is canceled; all others continue normally.

The value in *PropertyName* specifies the particular asynchronous data read request to cancel, and should match the value given in a previous **AsyncRead** method invocation. If *PropertyName* is not given, then the last **AsyncRead** method invocation that did not give a *PropertyName* will be canceled.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: RDO Data Control

Visual Studio 6.0

CancelBatch Method (Remote Data)

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

Cancels all uncommitted changes in the local cursor (used in batch mode).

Syntax

object.**CancelBatch** (*SingleRow*)

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>SingleRow</i>	A Boolean value that is True if the cancel action is done only for the current row, or False if the cancel action applies to all rows in the batch. Default is False.

Remarks

The **CancelBatch** method cancels all uncommitted changes in the local cursor, and reverts the data back to the state it was when originally fetched from the database. Note that this method does not refresh the data by re-querying the server like the **Refresh** method does instead it just discards changes made in the local cursor that have not already been sent in a batch update operation.

When you use the **CancelUpdate** method, only the current rows changes are rolled back to the state prior to execution of the last **Update** method.

Batch updates are used whenever you open a connection using the Client Batch cursor library (**rdUseClientBatch**). In this case, each time you use the **Update** or **UpdateRow** methods, the local **rdoResultset** is updated, but the base database tables are not changed. The **BatchUpdate** method is used to update the base database table(s) with any information changed since the **rdoResultset** was last created or synchronized with the **BatchUpdate** command.

The **BatchUpdate** method updates the **BatchCollisionRows** property to include a bookmark for each row that failed to update collided with an existing row that has data more current than the **rdoResultset** object as it existed when first read. The **BatchCollisionCount** property indicates how many collisions occurred during the batch update process.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: RDO Data Control

Visual Studio 6.0

CancelUpdate Method (Remote Data)

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

Cancels any pending updates to an **rdoResultset** object.

Syntax

object.**CancelUpdate**

The *object* placeholder represents an [object expression](#) that evaluates to an object in the Applies To list.

Remarks

The **CancelUpdate** method flushes the [copy buffer](#) and cancels any pending updates from an **Edit** or **AddNew** operation. For example, if a user invokes the **Edit** or **AddNew** method and hasnt yet invoked the **Update** method, **CancelUpdate** cancels any changes made after **Edit** or **AddNew** was invoked. Any information in the copy buffer is lost that is, any changes made to the [row](#) after the **Edit** or **AddNew** methods are invoked, are flushed.

Use the **EditMode** property to determine if there is a pending operation that can be canceled.

If the **CancelUpdate** method is used before using the **Edit** or **AddNew** methods or when the **EditMode** property is set to **rdEditNone**, the method is ignored.

Note Using the **CancelUpdate** method has the same effect as moving to another row without using the **Update** method, except that the [current row](#) doesnt change, and various properties, such as **BOF** and **EOF**, arent updated.

© 2018 Microsoft

Visual Basic: RDO Data Control

AddNew, Update, CancelUpdate Method Example

The following example illustrates use of the **AddNew** method to add new rows to a base table. This example assumes that you have read-write access to the table, that the column data provided meets the rules and other constraints associated with the table, and there is a unique index on the table. The data values for the operation are taken from three **TextBox** controls on the form. Note that the unique key for this table is not provided here as it is provided automatically it is an *identity* column.

```
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 AddNewJob_Click()
    On Error GoTo ANEH

    With rs
        .AddNew
        !job_desc = JobDescription
        !min_lvl = MinLevel
        !max_lvl = MaxLevel
        .Update
    End With
    Exit Sub

UpdateFailed:
    MsgBox "Update did not suceed."
    rs.CancelUpdate
    Exit Sub
A
NEH:
    Debug.Print Err, Error
    For Each er In rdoErrors
        Debug.Print er
    Next
    Resume UpdateFailed

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 = "JobsQuery"
    .SQL = "Select * from Jobs"
```

```
.RowsetSize = 1  
Set .ActiveConnection = cn  
Set rs = .OpenResultset(rdOpenKeyset, _  
    rdConcurRowver)  
Debug.Print rs.Updatable  
End With  
  
Exit Sub  
End Sub
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

CanPropertyChange Method

See Also Example [Applies To](#)

Asks the container if a property bound to a data source can have its value changed. The **CanPropertyChange** method is most useful if the property specified in *PropertyName* is bound to a data source.

Syntax

object.**CanPropertyChange** *PropertyName*

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>PropertyName</i>	A string expression that represents a name of the property that the control is requesting to change.

Return values

The possible return values for **CanPropertyChange** are:

Setting	Description
True	The property specified in <i>PropertyName</i> can be changed at this time.
False	The property specified in <i>PropertyName</i> cannot be changed at this time; the container has the bound data table open as read only. Do not set the property value; doing so may cause errors in some control containers.

Remarks

The control should always call **CanPropertyChange** before changing the value of a property that can be data-bound.

Note At present, **CanPropertyChange** always returns **True** in Visual Basic, even if the bound field is read-only in the data source. Visual Basic doesn't raise an error when the control attempts to change a read-only field; it just doesn't update the data source.

As an example, the following code shows how the **CanPropertyChange** method is used:

```
Public Property Let Address(ByVal cValue As String)
    If CanPropertyChange("Address") Then
        m_Address = cValue
        PropertyChanged "Address"
    End If
End Property
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: DataGrid Control

Visual Studio 6.0

CaptureImage Method

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

Returns a captured image of the grid's display in its current state.

Syntax

object.**CaptureImage**

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

Remarks

Use the **CaptureImage** method to retrieve a snapshot of the grid.

The following code uses the **CaptureImage** method to assign a snapshot of a **DataGrid** control to a **PictureBox** control.

```
Picture1.Picture = DataGrid1.CaptureImage
```

Note The **CaptureImage** method retrieves the image as a metafile typed image. Therefore, the image will resize to the size of its container.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: DataGrid Control

Visual Studio 6.0

CellText Method

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

Returns a formatted text value from a **DataGrid** control cell. Doesn't support named arguments.

Syntax

object.**CellText** *bookmark*

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>bookmark</i>	Required. A string expression that represents a row in the DataGrid control.

Remarks

The **CellText** method returns a formatted string representation of the data in the current column for the row specified by the *bookmark* value. Using the **CellText** method is similar to accessing the **Text** property, except you can select a specific row from which to retrieve the value.

The value returned by the **CellText** method is derived from the **Text** property by applying the formatting as specified by the **NumberFormat** property of the **Column** object.

When using the **CellText** method, use the **Columns** collection to specify the specific column of the **DataGrid** control and set the *bookmark* argument to a specific row.

Using the **CellText** method to extract information from a cell doesn't affect the current selection.

Visual Basic: DataGrid Control

CellText Method Example

This examples gets information from the top and bottom rows and displays it in a label.

```
Sub DataGrid1_Scroll (Cancel As Integer)
    Dim TopRow, BottomRow
    TopRow = DataGrid1.Columns(1).CellText(DataGrid1.FirstRow)
    BottomRow = DataGrid1.Columns(1).CellText(DataGrid1.RowBookmark _
        (DataGrid1.VisibleRows - 1))
    Label1.Caption = "Records " & TopRow & " to " & _
        BottomRow & " are currently displayed."
End Sub
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: DataGrid Control

Visual Studio 6.0

CellValue Method

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

Returns a raw data value in a column for a specified row in a **DataGrid** control. Doesn't support named arguments.

Syntax

object.**CellValue** *bookmark*

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>bookmark</i>	Required. A string expression that contains the unformatted data stored in a selected DataGrid control cell.

Remarks

When using the **CellValue** method, use the **Columns** collection to specify the specific column of the **DataGrid** control and set the *bookmark* argument to a specific row.

Using the **CellValue** method returns the same value as the **Value** property setting of the current **Column** object, except that you can specify a specific row in the **DataGrid** control to reference.

Using the **CellValue** method to extract information from a cell doesn't affect the current selection.

Visual Basic: DataGrid Control

CellValue Method Example

This example retrieves all the values in a given column from the selected range of rows and loads them into an array for later use.

```
Sub Command1_Click ()
    Dim I
    ReDim CalcArray (0 to DataGrid1.SelBookmarks.Count - 1)
    For I = 0 to DataGrid1.SelBookmarks.Count - 1
        ' Puts the value of the current row in the selected row
        ' array into corresponding CalcArray cell.
        CalcArray(I) = _
            DataGrid1.Columns(1).CellValue(DataGrid1.SelBookmarks(I))
    Next I
End Sub
```

© 2018 Microsoft

Visual Basic: DataGrid Control

CellValue Method Example

This example retrieves all the values in a given column from the selected range of rows and loads them into an array for later use.

```
Sub Command1_Click ()
    Dim I
    ReDim CalcArray (0 to DataGrid1.SelBookmarks.Count - 1)
    For I = 0 to DataGrid1.SelBookmarks.Count - 1
        ' Puts the value of the current row in the selected row
        ' array into corresponding CalcArray cell.
        CalcArray(I) = _
            DataGrid1.Columns(1).CellValue(DataGrid1.SelBookmarks(I))
    Next I
End Sub
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

Circle Method

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

Draws a circle, ellipse, or arc on an object.

Syntax

object.**Circle** [**Step**] (*x*, *y*), *radius*, [*color*, *start*, *end*, *aspect*]

The **Circle** method syntax has the following object qualifier and parts:

Part	Description
<i>object</i>	Optional. Object expression that evaluates to an object in the Applies To list. If <i>object</i> is omitted, the Form with the focus is assumed to be <i>object</i> .
Step	Optional. Keyword specifying that the center of the circle, ellipse, or arc is relative to the current coordinates given by the CurrentX and CurrentY properties of <i>object</i> .
(<i>x</i> , <i>y</i>)	Required. Single values indicating the coordinates for the center point of the circle, ellipse, or arc. The ScaleMode property of <i>object</i> determines the units of measure used.
<i>radius</i>	Required. Single value indicating the radius of the circle, ellipse, or arc. The ScaleMode property of <i>object</i> determines the unit of measure used.
<i>color</i>	Optional. Long integer value indicating the RGB color of the circle's outline. If omitted, the value of the ForeColor property is used. You can use the RGB function or QBColor function to specify the color.
<i>start</i> , <i>end</i>	Optional. Single-precision values. When an arc or a partial circle or ellipse is drawn, <i>start</i> and <i>end</i> specify (in radians) the beginning and end positions of the arc. The range for both is 2 pi radians to 2 pi radians. The default value for <i>start</i> is 0 radians; the default for <i>end</i> is 2 * pi radians.
<i>aspect</i>	Optional. Single-precision value indicating the aspect ratio of the circle. The default value is 1.0, which yields a perfect (non-elliptical) circle on any screen.

Remarks

To fill a circle, set the **FillColor** and **FillStyle** properties of the object on which the circle or ellipse is drawn. Only a closed figure can be filled. Closed figures include circles, ellipses, or pie slices (arcs with radius lines drawn at both ends).

When drawing a partial circle or ellipse, if *start* is negative, **Circle** draws a radius to *start*, and treats the angle as positive; if *end* is negative, **Circle** draws a radius to *end* and treats the angle as positive. The **Circle** method always draws in a counter-

clockwise (positive) direction.

The width of the line used to draw the circle, ellipse, or arc depends on the setting of the **DrawWidth** property. The way the circle is drawn on the background depends on the setting of the **DrawMode** and **DrawStyle** properties.

When drawing pie slices, to draw a radius to angle 0 (giving a horizontal line segment to the right), specify a very small negative value for *start*, rather than zero.

You can omit an argument in the middle of the syntax, but you must include the argument's comma before including the next argument. If you omit an optional argument, omit the comma following the last argument you specify.

When **Circle** executes, the **CurrentX** and **CurrentY** properties are set to the center point specified by the arguments.

This method cannot be used in an **WithEnd With** block.

© 2018 Microsoft

Visual Basic Reference

Circle Method Example

This example uses the **Circle** method to draw a number of concentric circles in the center of a form. To try this example, paste the code into the General section of a form. Then press F5 and click the form.

```
Sub Form_Click ()
    Dim CX, CY, Radius, Limit    ' Declare variable.
    ScaleMode = 3    ' Set scale to pixels.
    CX = ScaleWidth / 2    ' Set X position.
    CY = ScaleHeight / 2    ' Set Y position.
    If CX > CY Then Limit = CY Else Limit = CX
    For Radius = 0 To Limit    ' Set radius.
        Circle (CX, CY), Radius, RGB(Rnd * 255, Rnd * 255, Rnd * 255)
    Next Radius
End Sub
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

Clear Method (ActiveX Controls)

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

Removes all objects in a collection.

Syntax

object.**Clear**

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

Remarks

To remove only one object from a collection, use the **Remove** method.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

Clear Method (Clipboard, ComboBox, ListBox)

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

Clears the contents of a **ListBox**, **ComboBox**, or the system Clipboard.

Syntax

object.**Clear**

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

Remarks

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

© 2018 Microsoft

Visual Basic Reference

Clear Method Example

This example uses the **Clear** method to clear all items 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
```

This example uses the **Clear** method to clear the **Clipboard** object. 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_Click ()
    Const CF_BITMAP = 2 ' Define bitmap format.
    Dim Msg ' Declare variable.
    On Error Resume Next ' Set up error handling.
    Msg = "Choose OK to load a bitmap onto the Clipboard."
    MsgBox Msg ' Display message.
    Clipboard.Clear ' Clear Clipboard.
    Clipboard.SetData LoadPicture("PAPER.BMP") ' Get bitmap.
    If Err Then
        Msg = "Can't find the .BMP file."
        MsgBox Msg ' Display error message.
        Exit Sub
    End If
    Msg = "A bitmap is now on the Clipboard. Choose OK to copy "
    Msg = Msg & "the bitmap from the Clipboard to the form."
    MsgBox Msg ' Display message.
    Picture = Clipboard.GetData() ' Copy from Clipboard.
    Msg = "Choose OK to clear the picture."
    MsgBox Msg ' Display message.
    Picture = LoadPicture() ' Clear picture.
End Sub
```

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

Clear Method (DataObject Object)

See Also Example [Applies To](#)

Deletes the contents of the **DataObject** object.

Syntax

object.**Clear**

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

Remarks

This method is available only for component drag sources. If **Clear** is called from a component drop target, an error is generated.

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

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

Clear Method (MSHFlexGrid)

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

Clears the contents of the **MSHFlexGrid**. This includes all text, pictures, and cell formatting. This method does not affect the number of rows and columns within the **MSHFlexGrid**.

Syntax

object.**Clear**

The **Clear** method syntax has one part:

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

Remarks

To remove cells instead of just clearing them, use the **RemoveItem** method on each row to be removed.

© 2018 Microsoft

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Clear Method (MSHFlexGrid) Example

The following code places "Flex" into the current cell whenever the user clicks on a cell. When the user double-clicks, it clears the **MSHFlexGrid**. To run the program, press F5.

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

```
Private Sub Form1_Load ()
    MSHFlexGrid1.Rows =8
    MSHFlexGrid1.Cols =5
End Sub

Private Sub MSHFlexGrid1_Click ()
    ' Put text in current cell.
    MSHFlexGrid1.Text ="Flex"
End Sub

Private Sub MSHFlexGrid1_DblClick ()
    MSFlexGrid1.Clear
End Sub
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: RDO Data Control

Visual Studio 6.0

Clear Method (Remote Data)

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

Clears all members from the **rdoErrors** collection.

Syntax

object.**Clear**

The *object* placeholder represents an [object expression](#) that evaluates to an object in the Applies To list.

Remarks

Use this method to remove all **rdoError** objects from the **rdoErrors** collection.

Generally, it is unnecessary to clear the **rdoErrors** collection because it is automatically cleared when the first error occurs after initiating a new operation. Use the **Clear** method in cases where you need to clear the **rdoErrors** collection manually, for example if you wish to clear errors that have already been processed.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic for Applications Reference

Visual Studio 6.0

Clear Method

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

Clears all [property](#) settings of the **Err** object.

Syntax

object.**Clear**

The *object* is always the **Err** object.

Remarks

Use **Clear** to explicitly clear the **Err** object after an error has been handled, for example, when you use deferred error handling with **On Error Resume Next**. The **Clear** method is called automatically whenever any of the following [statements](#) is executed:

- Any type of **Resume** statement
- **Exit Sub**, **Exit Function**, **Exit Property**
- Any **On Error** statement

Note The **On Error Resume Next** construct may be preferable to **On Error GoTo** when handling errors generated during access to other objects. Checking **Err** after each interaction with an object removes ambiguity about which object was accessed by the code. You can be sure which object placed the error code in **Err.Number**, as well as which object originally generated the error (the object specified in **Err.Source**).

© 2018 Microsoft

Visual Basic for Applications Reference

Clear Method Example

This example uses the **Err** object's **Clear** method to reset the numeric properties of the **Err** object to zero and its string properties to zero-length strings. If **Clear** were omitted from the following code, the error message dialog box would be displayed on every iteration of the loop (after an error occurs) whether or not a successive calculation generated an error. You can single-step through the code to see the effect.

```
Dim Result(10) As Integer    ' Declare array whose elements
                             ' will overflow easily.
Dim indx
On Error Resume Next        ' Defer error trapping.
Do Until indx = 10
    ' Generate an occasional error or store result if no error.
    Result(indx) = Rnd * indx * 20000
    If Err.Number <> 0 Then
        MsgBox Err, , "Error Generated: ", Err.HelpFile, Err.HelpContext
        Err.Clear           ' Clear Err object properties.
    End If
    indx = indx + 1
Loop
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: DataGrid Control

Visual Studio 6.0

ClearFields Method

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

Restores the default grid layout.

Syntax

object.**ClearFields**

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

Remarks

The **ClearFields** method restores the default grid layout (with two blank columns) so that subsequent ReBind operations will automatically derive new column bindings from the (possibly changed) data source. You can cancel the grid's automatic layout behavior by invoking the **HoldFields** method.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: Windows Controls

Visual Studio 6.0

ClearSel Method

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

Clears the current selection of a **Slider** control.

Syntax

object.**ClearSel**

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

Remarks

This method sets the **SelStart** property to the value of the **Value** property and sets the **SelLength** property to 0.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: DataGrid Control

Visual Studio 6.0

ClearSelCols Method

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

Deselects all columns in a split. If no columns are selected, then this method does nothing.

Syntax

object.**ClearSelCols**

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

Remarks

If a grid contains multiple splits, then invoking its **ClearSelCols** method has the same effect as invoking the **ClearSelCols** method for the current split. The index of the current split is available through the **DataGrid** control's **Split** property.

Use the **SelStartCol** and **SelEndCol** properties to determine the current column selection range for a split.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

ClearStructure Method (MSHFlexGrid)

SeeAlso Example [Applies To](#)

Clears any mapping information from the **MSHFlexGrid** that regards the order and name of the displayed columns. This information may have been set at design time using the **Band** tab of the Property Pages dialog box or the Retrieve Structure command on the shortcut menu.

This method enables you to reset the **MSHFlexGrid** to a known state. If the user programmatically changes the underlying source of the **MSHFlexGrid** (and that source has a different structure), execute this method to properly display the data.

Syntax

object.**ClearStructure**

The **ClearStructure** method syntax has one part:

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

Remarks

If the structure is changed without clearing the mapping information, the **MSHFlexGrid** attempts to display the new source with the existing mapping information. Only columns specified in the mapping information display data in the **MSHFlexGrid**. Other columns specified in the mapping information display but dont show any data.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: Windows Controls

Visual Studio 6.0

Close Method (Animation Control)

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

The **Close** method causes the **Animation** control to close the currently open AVI file. If there was no file loaded, **Close** does nothing, and no error is generated

Syntax

object.**Close**

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

Remarks

To stop a file from playing, use the **Stop** method. However, if the **Autoplay** property is set to **True**, set **Autoplay** to **False** to stop the file from playing.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic for Applications Reference

Visual Studio 6.0

Close Method

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

Description

Closes an open **TextStream** file.

Syntax

object.**Close**

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

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

Close Method (OLE Container)

See Also Example [Applies To](#)

Closes an object and terminates the connection to the application that provided the object.

Syntax

object.**Close**

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

Remarks

This method applies only to embedded objects and is equivalent to closing the object. It has no effect on linked objects.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: RDO Data Control

Visual Studio 6.0

Close Method (Remote Data)

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

Closes an open [remote data object](#).

Syntax

object.Close

The *object* placeholder represents an [object expression](#) that evaluates to an object in the Applies To list.

Remarks

Closing an open object removes it from the collection of like objects except for the **rdoConnection** object. For example, using the **Close** method on an **rdoResultset** removes it from the **rdoResultsets** collection. However, using the **Close** method on the **rdoConnection** object, simply closes and discards any subordinate objects (like **rdoResultset** or **rdoQuery** objects) but does not remove it from the **rdoConnections** collection.

Closing the **rdoConnection** object also releases its parent **rdoEnvironment** object. Any attempt to close the [default environment](#) **rdoEnvironments(0)** is ignored. Unlike [DAO](#), RDO collection members cannot be removed with the **Delete** method.

If you try to close an **rdoConnection** object while any **rdoResultset** objects are open, or if you try to close an **rdoEnvironment** object while any **rdoConnection** objects belonging to that specific **rdoEnvironment** are open, those **rdoResultset** objects are closed and any pending updates or edits are rolled back.

If the **rdoConnection** object is defined outside the [scope](#) of the procedure, and you exit the procedure without closing it, the **rdoConnection** object remains open until it is explicitly closed or the module in which it is defined is out of scope. Any **rdoResultset** or **rdoQuery** objects that are opened against the **rdoConnection** remain open until explicitly closed. Once all [result sets](#) are closed on an **rdoConnection** that is no longer in scope, the **rdoConnection** is closed.

If *object* is already closed when you use **Close**, a trappable error is triggered.

Note Using the **Close** method against an executing [action query](#) might have unpredictable results. If the query is performing an operation that affects a number of rows, some of the rows might be changed, while others are not. For example, if you execute an action query containing an [SQL UPDATE](#) statement and use the **Close** method before the operation is complete, an indeterminate number of rows are updated leaving others unchanged. If you intend to use the **Close** method against this type of action query, it is recommended that you use transaction methods to roll back or commit partially completed operations.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Extensibility Reference

Visual Studio 6.0

Close Method (VBA Add-In Object Model)

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

Closes and destroys a window.

Syntax

object.**Close**

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

Remarks

The following types of windows respond to the **Close** method in different ways:

- For a window that is a code pane, **Close** destroys the code pane.
- For a window that is a designer, **Close** destroys the contained designer.
- For windows that are always available on the **View** menu, **Close** hides the window.

© 2018 Microsoft

Visual Basic Extensibility Reference

Close Method Example

The following example uses the **Close** method to close a specified member of the **Windows** collection.

```
Application.VBE.Windows(9).Close
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: Winsock Control

Visual Studio 6.0

Close Method (Winsock Control)

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

Closes a TCP connection or a listening socket for both client and server applications.

Syntax

object.**Close**

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

Arguments

None

Return Value

Void

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

Cls Method

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

Clears graphics and text generated at [run time](#) from a **Form** or **PictureBox**.

Syntax

object.Cls

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list. If *object* is omitted, the **Form** with the [focus](#) is assumed to be *object*.

Remarks

Cls clears text and graphics generated at run time by graphics and printing statements. Background [bitmaps](#) set using the **Picture** property and controls placed on a **Form** at design time aren't affected by **Cls**. Graphics and text placed on a **Form** or **PictureBox** while the **AutoRedraw** property is set to **True** aren't affected if **AutoRedraw** is set to **False** before **Cls** is invoked. That is, you can maintain text and graphics on a **Form** or **PictureBox** by manipulating the **AutoRedraw** property of the object you're working with.

After **Cls** is invoked, the **CurrentX** and **CurrentY** properties of *object* are reset to 0.

© 2018 Microsoft

Visual Basic Reference

Cls Method Example

This example uses the **Cls** method to delete printed information from a form. 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_Click ()
    Dim Msg      ' Declare variable.
    AutoRedraw = -1    ' Turn on AutoRedraw.
    ForeColor = QBColor(15)    ' Set foreground to white.
    BackColor = QBColor(1)     ' Set background to blue.
    FillStyle = 7    ' Set diagonal crosshatch.
    Line (0, 0)-(ScaleWidth, ScaleHeight), , B    ' Put box on form.
    Msg = "This is information printed on the form background."
    CurrentX = ScaleWidth / 2 - TextWidth(Msg) / 2    ' Set X position.
    CurrentY = 2 * TextHeight(Msg)    ' Set Y position.
    Print Msg    ' Print message to form.
    Msg = "Choose OK to clear the information and background "
    Msg = Msg & "pattern just displayed on the form."
    MsgBox Msg    ' Display message.
    Cls    ' Clear form background.
End Sub
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: DataGrid Control

Visual Studio 6.0

ColContaining Method

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

Returns the **ColIndex** value of the **DataGrid** control column containing the specified coordinate (X) value. Doesn't support named arguments.

Syntax

object.ColContaining *coordinate*

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>coordinate</i>	Required. A single numeric expression that defines a horizontal coordinate (X value) based on the coordinate system of the container.

Remarks

The **ColContaining** method returns a number that corresponds to one of the column indexes of the control specified by *object*. This number ranges from 0 to 1 less than the setting of the **Count** property of the **Columns** collection (0 to **Columns.Count** - 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 column of the **DataGrid** control.

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

Note The **ColContaining** method returns the **ColIndex** of the column indicated, not the visible column. If *coordinate* falls in the first visible column, but two columns have been scrolled off the left side of the control, the **ColContaining** method returns 2.

© 2018 Microsoft

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
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

CollapseAll Method (MSHFlexGrid)

[SeeAlso](#) [Example](#) [Applies To](#)

Collapses all rows of the specified band within the **MSHFlexGrid**.

Syntax

object.**CollapseAll**(*number*)

The **CollapseAll** 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>	Optional. A Long value that specifies the band that contains the rows to collapse.

This documentation is archived and is not being maintained.

Visual Basic: RDO Data Control

Visual Studio 6.0

ColumnSize Method (Remote Data)

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

Returns the number of bytes in an **rdoColumn** object with a [data type](#) of **rdTypeLONGVARBINARY** or **rdTypeLONGVARCHAR**.

Syntax

varname = *object* ! *column*.**ColumnSize**()

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

Part	Description
<i>varname</i>	The name of a Long or Variant variable.
<i>object</i>	An object expression that evaluates to the rdoResultset object containing the rdoColumns collection.
<i>column</i>	The name of an rdoColumn object whose ChunkRequired property is set to True .

Remarks

Depending on the driver being used, the **ColumnSize** method either returns the size of a binary large object (BLOB) column, or -1 if the size is not available. If the BLOB column size is not available, you can still use the **GetChunk** method to read chunks of data from your BLOB column. The last block has been fetched when the value returned by **GetChunk** is smaller than the size requested (for binary data), at least two bytes smaller than your buffer (for character data), or returns a NULL value.

When working with data types that span multiple database [pages](#), you should use the *chunk* methods to manage the data but this is not an absolute requirement. You should also use the **GetChunk** and **AppendChunk** methods to manage *chunk* data when the **ChunkRequired** property is **True**. Note that when the size of BLOB data columns is smaller than the **BindThreshold**, it is not necessary to use the chunk methods.

Use the **ColumnSize** method to determine the size of *chunk* columns.

Because the size of a *chunk* data column can exceed 64K, you should assign the value returned by the **GetChunk** method to a variable large enough to store the data returned based on the size returned by the **ColumnSize** method.

Note To determine the size of a non-*chunk* **rdoColumn** object, use the **Size** property.

Visual Basic: RDO Data Control

AppendChunk, GetChunk Method Example

This example illustrates use of the **AppendChunk** and **GetChunk** methods to write page-based binary large object (BLOB) data to a remote data source. The code expects a table with a char, text, and image field named *Chunks*. To create this table, submit the following as an action query against your test database:

```
CREATE TABLE Chunks (ID integer identity NOT NULL, PName char(10) NULL,
Description TEXT NULL,
Photo IMAGE NULL)
CREATE UNIQUE INDEX ChunkIDIndex on Chunks(ID)
```

Once the table is created, you will need to locate one or more .BMP or other suitable graphics images that can be loaded by the **PictureBox** control.

```
'
Option Explicit
Dim en As rdoEnvironment
Dim Qd As rdoQuery
Dim Cn As rdoConnection
Dim Rs As rdoResultset
Dim SQL As String
Dim DataFile As Integer, Fl As Long, Chunks As Integer
Dim Fragment As Integer, Chunk() As Byte, I As Integer
Const ChunkSize As Integer = 16384

Private Sub Form_Load()
Set en = rdoEnvironments(0)
Set Cn = en.OpenConnection(dsname:="", _
    Connect:="UID=;PWD=;DATABASE=WorkDB;" _
    & "Driver={SQL Server};SERVER=Betav486", _
    prompt:=rdDriverNoPrompt)
Set Qd = Cn.CreateQuery("TestChunk", "Select * from
    Chunks Where PName = ?")
End Sub

Private Sub LoadFromFile_Click()
'
' Locates a file and sets the Filename to this file.
'
With CommonDialog1
    .Filter = "Pictures(*.bmp;*.ico)|*.bmp;*.ico"
    .ShowOpen
    FileName = .FileName
End With
End Sub

Private Sub ReadFromDB_Click()
If Len(NameWanted) = 0 Then _
    NameWanted = InputBox("Enter name wanted", "Animal")
Qd(0) = NameWanted
Set Rs = Qd.OpenResultset(rdOpenKeyset, _
    rdConcurRowver)
```

```

If Rs Is Nothing Or Rs.Updatable = False Then
    MsgBox "Cant open or write to result set"
    Exit Sub
End If
If Rs.EOF Then
    MsgBox "Cant find picture by that name"
    Exit Sub
End If
Description = Rs!Description
DataFile = 1
Open "pictemp" For Binary Access Write As DataFile
Fl = Rs!Photo.ColumnSize
Chunks = Fl \ ChunkSize
Fragment = Fl Mod ChunkSize
ReDim Chunk(Fragment)
Chunk() = Rs!Photo.GetChunk(Fragment)
Put DataFile, , Chunk()
For I = 1 To Chunks
    ReDim Buffer(ChunkSize)
    Chunk() = Rs!Photo.GetChunk(ChunkSize)
    Put DataFile, , Chunk()
Next I
Close DataFile
FileName = "pictemp"
End Sub

Private Sub SaveToDB_Click()
If Len(NameWanted) = 0 Then _
    NameWanted = InputBox("Enter name for this" _
        & " picture", "Animal")
    Qd(0) = NameWanted
    Set Rs = Qd.OpenResultset(rdOpenKeyset, _
        rdConcurRowver)
If Rs Is Nothing Or Rs.Updatable = False Then
    MsgBox "Cant open or write to result set"
    Exit Sub
End If
If Rs.EOF Then
    Rs.AddNew
    Rs!PName = NameWanted
If Description = "" Then _
    Description = InputBox("Describe the picture", _
        "Dont care")
    'Rs!Description = Description
Else
    Rs.Edit
End If
DataFile = 1
Open FileName For Binary Access Read As DataFile
Fl = LOF(DataFile) ' Length of data in file
If Fl = 0 Then Close DataFile: Exit Sub
Chunks = Fl \ ChunkSize
Fragment = Fl Mod ChunkSize
Rs!Photo.AppendChunk Null
ReDim Chunk(Fragment)
Get DataFile, , Chunk()
Rs!Photo.AppendChunk Chunk()
ReDim Chunk(ChunkSize)
For I = 1 To Chunks
    Get DataFile, , Chunk()
    Rs!Photo.AppendChunk Chunk()

```

```
Next I  
Close DataFile  
Rs.Update  
End Sub
```

```
Private Sub FileName_Change()  
Picture1.Picture = LoadPicture(FileName)  
End Sub
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: MAPI Controls

Visual Studio 6.0

Compose Method

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

Composes a message.

Syntax

object.Compose

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

Remarks

This method clears all the components of the compose buffer, and sets the **MsgIndex** property to -1.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: Windows Controls

Visual Studio 6.0

ComputeControlSize Method

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

Returns the width and height of a **MonthView** control for a given number of rows and columns.

Syntax

object.**ComputeControlSize**(Rows, Columns, Width, Height)

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>Rows</i>	Sets an integer specifying the number of rows.
<i>Columns</i>	Sets an integer specifying the number of columns.
<i>Width</i>	Returns a single-precision number that is the width of the prospective control.
<i>Height</i>	Returns a single-precision number that is the height of the prospective control.

Remarks

When the size of the **MonthView** control changes by resetting the **MonthColumns** or **MonthRows** properties, use the **ComputeControlSize** method to calculate the size of the control before the change, and resize the form accordingly.

To use the **ComputeControlSize** method, first declare two variables as type **Single**. Then invoke the method with the variables as the *Width* and *Height* arguments.

Visual Basic: Windows Controls

ComputeControlSize Method Example

The example uses the **ComputeControlSize** method to calculate how large a **MonthView** control will be after increasing its size using the **MonthRows** and **MonthColumns** properties. To try the example, place a **MonthView** control on a form and paste the code into the Declarations section of the code module. Start the project and double-click the form.

```
Private Sub Form_Load()  
    ' Set the Top and Left properties of the control.  
    With MonthView1  
        .Left = 200 ' Assuming ScaleMode = Twip  
        .Top = 400  
    End With  
End Sub  
  
Private Sub Form_DblClick()  
    Dim sWidth As Single  
    Dim sHeight As Single  
  
    With MonthView1 ' Compute the control size and add columns and rows.  
        .ComputeControlSize 3, 4, sWidth, sHeight  
        .MonthColumns = 4  
        .MonthRows = 3  
    End With  
  
    ' Resize the control using the values from the method.  
    Me.Width = MonthView1.Left + sWidth + 500  
    Me.Height = MonthView1.Top + sHeight + 500  
End Sub
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: Winsock Control

Visual Studio 6.0

Connect Method

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

Requests a connection to a remote computer.

Syntax

object.**Connect** *remoteHost*, *remotePort*

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>remoteHost</i>	Required. The name of the remote computer to connect to.
<i>remotePort</i>	The port of the remote computer to connect to.

Return Value

None

Remarks

You must invoke the Connect method when attempting to establish a TCP connection.

This documentation is archived and is not being maintained.

Visual Basic: MAPI Controls

Visual Studio 6.0

Copy Method (MAPIMessages Control)

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

Copies the currently indexed message to the compose buffer.

Syntax

object.Copy

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

Remarks

This method sets the **MsgIndex** property to -1.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

Copy Method (OLE Container Control)

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

Copies the object within an **OLE** container control to the system Clipboard.

Syntax

object.**Copy**

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

Remarks

When you copy an object onto the system Clipboard, all the data and link information associated with the object is placed on the system Clipboard. You can copy both linked and embedded objects onto the system Clipboard.

You can use this method to support an Edit Copy command on a menu.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Extensibility Reference

Visual Studio 6.0

Copy Method

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

Copies the selected controls on the form to the Clipboard.

Syntax

object.**Copy**

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

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic for Applications Reference

Visual Studio 6.0

Copy Method

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

Description

Copies a specified file or folder from one location to another.

Syntax

object.**Copy** *destination*[, *overwrite*]

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

Part	Description
<i>object</i>	Required. Always the name of a File or Folder object.
<i>destination</i>	Required. Destination where the file or folder is to be copied. Wildcard characters are not allowed.
<i>overwrite</i>	Optional. Boolean value that is True (default) if existing files or folders are to be overwritten; False if they are not.

Remarks

The results of the **Copy** method on a **File** or **Folder** are identical to operations performed using **FileSystemObject.CopyFile** or **FileSystemObject.CopyFolder** where the file or folder referred to by *object* is passed as an argument. You should note, however, that the alternative methods are capable of copying multiple files or folders.

This documentation is archived and is not being maintained.

Visual Basic for Applications Reference

Visual Studio 6.0

CopyFile Method

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

Description

Copies one or more files from one location to another.

Syntax

object.**CopyFile** *source*, *destination*[, *overwrite*]

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

Part	Description
<i>object</i>	Required. The <i>object</i> is always the name of a FileSystemObject .
<i>source</i>	Required. Character string file specification, which can include wildcard characters, for one or more files to be copied.
<i>destination</i>	Required. Character string destination where the file or files from <i>source</i> are to be copied. Wildcard characters are not allowed.
<i>overwrite</i>	Optional. Boolean value that indicates if existing files are to be overwritten. If True , files are overwritten; if False , they are not. The default is True . Note that CopyFile will fail if <i>destination</i> has the read-only attribute set, regardless of the value of <i>overwrite</i> .

Remarks

Wildcard characters can only be used in the last path component of the *source* argument. For example, you can use:

```
FileSystemObject.CopyFile "c:\mydocuments\letters\*.doc", "c:\tempfolder\"
```

But you can't use:

```
FileSystemObject.CopyFile "c:\mydocuments\*\R1???97.xls", "c:\tempfolder"
```

If *source* contains wildcard characters or *destination* ends with a path separator (\), it is assumed that *destination* is an existing folder in which to copy matching files. Otherwise, *destination* is assumed to be the name of a file to create. In either case, three things can happen when an individual file is copied.

- If *destination* does not exist, *source* gets copied. This is the usual case.
- If *destination* is an existing file, an error occurs if *overwrite* is **False**. Otherwise, an attempt is made to copy *source* over the existing file.
- If *destination* is a directory, an error occurs.

An error also occurs if a *source* using wildcard characters doesn't match any files. The **CopyFile** method stops on the first error it encounters. No attempt is made to roll back or undo any changes made before an error occurs.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic for Applications Reference

Visual Studio 6.0

CopyFolder Method

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

Description

Recursively copies a folder from one location to another.

Syntax

object.**CopyFolder** *source*, *destination*[, *overwrite*]

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

Part	Description
<i>object</i>	Required. Always the name of a FileSystemObject .
<i>source</i>	Required. Character string folder specification, which can include wildcard characters, for one or more folders to be copied.
<i>destination</i>	Required. Character string destination where the folder and subfolders from <i>source</i> are to be copied. Wildcard characters are not allowed.
<i>overwrite</i>	Optional. Boolean value that indicates if existing folders are to be overwritten. If True , files are overwritten; if False , they are not. The default is True .

Remarks

Wildcard characters can only be used in the last path component of the *source* argument. For example, you can use:

```
FileSystemObject.CopyFolder "c:\mydocuments\letters\*", "c:\tempfolder\"
```

But you can't use:

```
FileSystemObject.CopyFolder "c:\mydocuments\*\*", "c:\tempfolder\"
```

If *source* contains wildcard characters or *destination* ends with a path separator (\), it is assumed that *destination* is an existing folder in which to copy matching folders and subfolders. Otherwise, *destination* is assumed to be the name of a folder to create. In either case, four things can happen when an individual folder is copied.


- If *destination* does not exist, the *source* folder and all its contents gets copied. This is the usual case.

- If *destination* is an existing file, an error occurs.
- If *destination* is a directory, an attempt is made to copy the folder and all its contents. If a file contained in *source* already exists in *destination*, an error occurs if *overwrite* is **False**. Otherwise, it will attempt to copy the file over the existing file.
- If *destination* is a read-only directory, an error occurs if an attempt is made to copy an existing read-only file into that directory and *overwrite* is **False**.

An error also occurs if a *source* using wildcard characters doesn't match any folders.

The **CopyFolder** method stops on the first error it encounters. No attempt is made to roll back any changes made before an error occurs.

© 2018 Microsoft



This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

Count Method

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

Returns the number of objects in a collection.

Syntax

object.**Count**

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

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: Windows Controls

Visual Studio 6.0

CreateDragImage Method

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

Creates a drag image using a dithered version of an object's associated image. This image is typically used in drag-and-drop operations.

Syntax

object.**CreateDragImage**

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

Remarks

The **CreateDragImage** method is typically used to assign an image to a **DragIcon** property at the start of a drag-and-drop operation.

© 2018 Microsoft

Visual Basic: Windows Controls

CreateDragImage Method Example

This example adds several **Node** objects to a **TreeView** control. After you select a **Node** object, you can drag it to any other **Node**. To try the example, place **TreeView** and **ImageList** controls on a form and paste the code into the form's Declaration section. Run the example and drag **Node** objects around to see the result.

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.

```
' Declare global variables.
Dim indrag As Boolean ' Flag that signals a Drag Drop operation.
Dim nodX As Object ' Item that is being dragged.

Private Sub Form_Load()
    ' Load a bitmap into an ImageList control.
    Dim imgX As ListImage
    Dim BitmapPath As String
    BitmapPath = "icons\mail\mail01a.ico" ' Change this to a valid path.
    Set imgX = imagelist1.ListImages.Add(, , LoadPicture(BitmapPath))

    ' Initialize TreeView control and create several nodes.
    TreeView1.ImageList = imagelist1
    Dim nodX As Node ' Create a tree.
    Set nodX = TreeView1.Nodes.Add(, , "Parent1", 1)
    Set nodX = TreeView1.Nodes.Add(, , "Parent2", 1)
    Set nodX = TreeView1.Nodes.Add(1, tvwChild, "Child 1", 1)
    Set nodX = TreeView1.Nodes.Add(1, tvwChild, "Child 2", 1)
    Set nodX = TreeView1.Nodes.Add(2, tvwChild, "Child 3", 1)
    Set nodX = TreeView1.Nodes.Add(2, tvwChild, "Child 4", 1)
    Set nodX = TreeView1.Nodes.Add(3, tvwChild, "Child 5", 1)
    nodX.EnsureVisible ' Expand tree to show all nodes.
End Sub

Private Sub TreeView1_MouseDown_
(Button As Integer, Shift As Integer, x As Single, y As Single)
    Set nodX = TreeView1.SelectedItem ' Set the item being dragged.
End Sub

Private Sub TreeView1_MouseMove _
(Button As Integer, Shift As Integer, x As Single, y As Single)
    If Button = vbLeftButton Then ' Signal a Drag operation.
        indrag = True ' Set the flag to true.
        ' Set the drag icon with the CreateDragImage method.
        TreeView1.DragIcon = TreeView1.SelectedItem.CreateDragImage
        TreeView1.Drag vbBeginDrag ' Drag operation.
    End If
End Sub

Private Sub TreeView1_DragDrop_
(Source As Control, x As Single, y As Single)
    If TreeView1.DropHighlight Is Nothing Then
        Set TreeView1.DropHighlight = Nothing
    End If
End Sub
```

```
    indrag = False
    Exit Sub
Else
    If nodX = TreeView1.DropHighlight Then Exit Sub
    Cls
    Print nodX.Text & " dropped on " & TreeView1.DropHighlight.Text
    Set TreeView1.DropHighlight = Nothing
    indrag = False
End If
End Sub

Private Sub TreeView1_DragOver(Source As Control, x As Single, y As Single, State As Integer)
    If indrag = True Then
        ' Set DropHighlight to the mouse's coordinates.
        Set TreeView1.DropHighlight = TreeView1.HitTest(x, y)
    End If
End Sub
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

CreateEmbed Method

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

Creates an embedded object. Doesn't support named arguments.

Syntax

object.**CreateEmbed** *sourcedoc*, *class*

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

Part	Description
<i>Object</i>	An object expression that evaluates to an object in the Applies To list.
<i>Sourcedoc</i>	Required. The filename of a document used as a template for the embedded object. Must be a zero-length string ("") if you don't specify a source document.
<i>Class</i>	Optional. The name of the class of the embedded object. Ignored if you specify a filename for <i>sourcedoc</i> .

Remarks

To view a list of valid class names available on your system, select a control, such as the **OLE** container control, select the **Class** property in the Properties window, and then click the builder button.

Note You don't need to set the **Class** and **SourceDoc** properties when using the **CreateEmbed** method to create an embedded object.

When you create a new object, the application associated with the class name (for example, Excel.exe) must be correctly registered with the operating system. (The application setup program should register the application correctly.)

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Extensibility Reference

Visual Studio 6.0

CreateEventProc Method

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

Creates an event procedure.

Syntax

object.**CreateEventProc**(*eventname*, *objectname*) **As Long**

The **CreateEventProc** syntax has these parts:

Part	Description
<i>object</i>	Required. An object expression that evaluates to an object in the Applies To list.
<i>eventname</i>	Required. A string expression specifying the name of the event you want to add to the module.
<i>objectname</i>	Required. A string expression specifying the name of the object that is the source of the event.

Remarks

Use the **CreateEventProc** method to create an event procedure. For example, to create an event procedure for the Click event of a **Command Button** control named Command1 you would use the following code, where CM represents a object of type **CodeModule**:

```
TextLocation = CM.CreateEventProc("Click", "Command1")
```

The **CreateEventProc** method returns the line at which the body of the event procedure starts. **CreateEventProc** fails if the arguments refer to a nonexistent event.

© 2018 Microsoft

Visual Basic Extensibility Reference

CreateEventProc Method Example

The following example uses the **CreateEventProc** method to create the Button_Click procedure.

```
Debug.Print Application.VBE.SelectVBComponents.CodeModule.CreateEventProc("Click", "Button")
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic for Applications Reference

Visual Studio 6.0

CreateFolder Method

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

Description

Creates a folder.

Syntax

object.**CreateFolder**(*foldername*)

The **CreateFolder** method has these parts:

Part	Description
<i>object</i>	Required. Always the name of a FileSystemObject .
<i>foldername</i>	Required. String expression that identifies the folder to create.

Remarks

An error occurs if the specified folder already exists.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

CreateLink Method

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

Creates a linked object from the contents of a file. Doesn't support named arguments.

Syntax

object.**CreateLink** *sourcedoc*, *sourceitem*

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

Part	Description
<i>Object</i>	An object expression that evaluates to an object in the Applies To list.
<i>Sourcedoc</i>	Required. The file from which the object is created.
<i>Sourceitem</i>	Optional. The data within the file to be linked in the linked object.

Remarks

If you specify values for the arguments of this method, those values override the settings of the **SourceDoc** and **SourceItem** properties. Those properties are updated to reflect the argument values when the method is invoked.

When an object is created with this method, the **OLE** container control displays an image of the file specified by the **SourceDoc** property. If the object is saved, only the link references are saved because the **OLE** container control contains only a [metafile](#) image of the data and no actual source data.

When you create a new object, the application associated with the class name (for example, Excel.exe) must be correctly registered with the operating system. (The application setup program should register the application correctly.)

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: RDO Data Control

Visual Studio 6.0

CreatePreparedStatement Method (Remote Data)

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

Creates a new **rdoPreparedStatement** object.

Syntax

Set *prepstmt* = *connection*.**CreatePreparedStatement**(*name*, *sqlstring*)

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

Part	Description
<i>prepstmt</i>	An object expression that evaluates to the rdoPreparedStatement object you want to create.
<i>connection</i>	An object expression that represents the open rdoConnection object.
<i>name</i>	A String that is the name of the new rdoPreparedStatement . This part is required, but may be an empty string ().
<i>sqlstring</i>	A Variant expression (a valid SQL statement) that defines the rdoPreparedStatement . This part is required, but you can provide an empty string if you do, you must define the rdoPreparedStatement by setting its SQL property before executing the new rdoPreparedStatement .

Remarks

Note Support for the **rdoPreparedStatement** object is provided in this version of Visual Basic to provide compatibility with previous versions. The **rdoQuery** object should be used as a direct replacement for this object. Because of this, it is also recommended that use of the **CreatePreparedStatement** method be discontinued in favor of the **CreateQuery** method.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: RDO Data Control

Visual Studio 6.0

CreateQuery Method (Remote Data)

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

Creates a new query object and adds it to the **rdoQueries** collection.

Syntax

object.**CreateQuery** *Name*, *SQLString*

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

Part	Description
<i>object</i>	An object expression that evaluates to an rdoConnection object
<i>Name</i>	Required. A string expression that evaluates to the name for the new object
<i>SQLString</i>	Optional. SQL query for the new prepared statement

Remarks

The **CreateQuery** method creates a new **rdoQuery** object for this connection and adds it to the **rdoQueries** collection. You can also declare stand-alone **rdoQuery** objects using the **Dim** statement as follows:

```
Dim myQuery as New rdoQuery
```

Stand-alone **rdoQuery** objects are not associated with a connection until you set the **ActiveConnection** property.

The **rdoQuery** corresponds to the ODBC prepared statement used to define a reusable SQL query that can contain parameters. You can execute the **rdoQuery** any number of times, and pass parameters that are substituted into the SQL statement before it is executed. Parameters are maintained in the **rdoParameters** collection. Generally, if you intend to execute a query more than once in your code, it is more efficient to use **rdoQuery** objects than to use the **Execute** or **OpenResultset** method on objects other than the **rdoQuery**.

The value passed for the **Name** parameter can be used with the **Item** method to locate the new object in its collection. If **Name** is not provided, the **rdoQuery** is appended to the **rdoQueries** collection, and the **rdoQuery** can be used by referencing the query variable or the **rdoQuery** objects ordinal value. If the object specified by name is already a member of the **rdoQueries** collection (including an empty string), a trappable error occurs. All **rdoQuery** objects are temporary they are discarded when the **rdoConnection** object is closed.

To remove an **rdoQuery** object from an **rdoQueries** collection, use the **Close** method on the **rdoQuery**.

The **SQLString** parameter is optional, but if not provided, you must set the **SQL** property of the resulting **rdoQuery** object before executing it.

Use the **Execute** method to run an SQL statement in an **rdoQuery** object that does not return rows (an action query). Use the **OpenResultset** method to run an **rdoQuery** that returns rows.

If there is an unpopulated **rdoResultset** pending on a data source that can only support a single operation on an **rdoConnection** object, you cannot create additional **rdoQuery** or **rdoResultset** objects, or use the **Refresh** method on the **rdoTable** object until the **rdoResultset** is flushed, closed, or fully populated. For example, when using SQL Server 4.2 as a data source, you cannot create an additional **rdoResultset** object until you move to the last row of the current **rdoResultset** object. To populate the result set, use the **MoreResults** method to move through all pending result sets, or use the **Cancel** or **Close** method on the **rdoResultset** to flush all pending result sets.

© 2018 Microsoft

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
```

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic for Applications Reference

Visual Studio 6.0

CreateTextFile Method

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

Description

Creates a specified file name and returns a **TextStream** object that can be used to read from or write to the file.

Syntax

object.**CreateTextFile**(*filename*[, *overwrite*[, *unicode*]])

The **CreateTextFile** method has these parts:

Part	Description
<i>object</i>	Required. Always the name of a FileSystemObject or Folder object.
<i>filename</i>	Required. String expression that identifies the file to create.
<i>overwrite</i>	Optional. Boolean value that indicates if an existing file can be overwritten. The value is True if the file can be overwritten; False if it can't be overwritten. If omitted, existing files are not overwritten.
<i>unicode</i>	Optional. Boolean value that indicates whether the file is created as a Unicode or ASCII file. The value is True if the file is created as a Unicode file; False if it's created as an ASCII file. If omitted, an ASCII file is assumed.

Remarks

The following code illustrates how to use the **CreateTextFile** method to create and open a text file:

```
Sub CreateAfile
    Set fs = CreateObject("Scripting.FileSystemObject")
    Set a = fs.CreateTextFile("c:\testfile.txt", True)
    a.WriteLine("This is a test.")
    a.Close
End Sub
```

If the *overwrite* argument is **False**, or is not provided, for a *filename* that already exists, an error occurs.

This documentation is archived and is not being maintained.

Visual Basic Extensibility Reference

Visual Studio 6.0

CreateToolWindow Method

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

Creates a new Tool window containing the indicated **UserDocument** object.

Syntax

object.**CreateToolWindow** (*AddInInst*, *ProgID*, *Caption*, *GuidPosition*, *DocObj*) **As Window**

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

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>AddInInst</i>	Required. An add-in instance variable representing an add-in in the development environment.
<i>ProgID</i>	Required. String representing the progID of the UserDocument object.
<i>Caption</i>	Required. String containing the window caption.
<i>GuidPosition</i>	Required. String containing a unique identifier for the window.
<i>DocObj</i>	Required. Object representing a UserDocument object. This object will be set in the call to this function.

This documentation is archived and is not being maintained.

Visual Basic: Windows Controls

Visual Studio 6.0

Customize Method

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

Invokes the Customize Toolbar dialog box which allows the end user to rearrange or hide **Button** objects on a **Toolbar** control.

Syntax

object.**Customize**

The *object* placeholder is an object expression that evaluates to a **Toolbar** control.

Remarks

The **Toolbar** control contains a built-in dialog box that allows the user to hide, display, or rearrange buttons on a toolbar. Double-clicking the toolbar calls the **Customize** method, which invokes the dialog box.

Use the **Customize** method when you wish to restrict the alteration of the toolbar. For example, the code below allows the user to customize the toolbar only if a password is given:

```
Private Sub Command1_Click()  
    If InputBox("Password:") = "Chorus&Line9" Then  
        Toolbar1.Customize    ' Invoke Customize method.  
    End If  
End Sub
```

To preserve the state of a **Toolbar** control, the **SaveToolbar** method writes to the Windows registry. You can restore a **Toolbar** control to a previous state using the **RestoreToolbar** method to read the information previously saved in the registry.

© 2018 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Extensibility Reference

Visual Studio 6.0

Cut Method

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

Deletes the selected controls from the form.

Syntax

object.**Cut**

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

© 2018 Microsoft