RemoteDataBar ActiveX Control

Properties Methods Events

RemoteDataBar provides access to data stored in a remote OLE DB (ADO) data source through bound controls. It communicates with a server machine over HTTP, HTTPS, DCOM (for a local area network), and any other protocols supported by Microsoft® Remote Data Service® (RDS). You don't need to know about RDS or how it works in order to use **RemoteDataBar**.

RemoteDataBar thus allows you to bind controls at design time (or run time) to a data source "over the web", using a development environment such as Microsoft Visual Basic®. This can be a tremendous advantage over other data source objects that either require a direct database connection over a local network (and consume expensive database resources), or can only be bound to controls on an HTML page.

Once the server machine is set up to support the communication, in order to access data you'll need to place an instance of the control on a form, set the **Server** property to the server machine's name or URL, and set the ADO connection string with the **Connect** property. Then you define the source of the data using the **RecordSource** property, which is a SQL SELECT statement if you're accessing a DBMS such as Oracle, SQL Server or Access.

Connect the **RemoteDataBar** control to a data-bound control such as the **DataGrid**, **DataCombo**, or **TextBox** control by setting the **DataSource** property to the **RemoteDataBar** control. For a **TextBox** you'll also need to set the **DataField** property to the appropriate database field.

At run time, you can change any of these properties to use a different server machine or database. You can also change the underlying recordset to a previously opened recordset with the **SourceRecordset** property. You have full access to the underlying recordset through the **Recordset** property.

Because **RemoteDataBar** is built on top of the Data Access Objects Client, you have access to its data services through the **Database** property.

Because the data access objects client components and **RemoteDataBar** are so closely tied in with ADO, you should have access to the ADO documentation to be able to fully understand many of the constants and parameters used and referred to in this help. If you are using Visual Basic as your development environment, then you should add "Microsoft ActiveX Data Objects 2.0 Library" in the "Project—References" dialog so that you have access to these constants.

bound control

A data-aware control that can provide access to a specific field or fields in a database through a Data control. A data-aware control is typically bound to a Data control through its DataSource, DataField and DataMember properties. When a Data control moves from one record to the next, all bound controls connected to the Data control change to display data from fields in the current record. When users change data in a bound control and then move to a different record, the changes are automatically saved in the database.

Data Access Objects Client Overview

See Also

The Data Access Objects Client component provides access to data stored in a remote OLE DB (ADO) data source. It communicates with a server machine over HTTP, HTTPS, DCOM (for a local area network), and any other protocols supported by Microsoft Remote Data Service (RDS). The client component comprises the **Database** object, the **Parameters** collection, and the **Parameter** object. You don't need to know about RDS or how it works in order to use these objects.

Because the Database object deals only with disconnected recordsets, no direct database connections are maintained by the client; thus, expensive database resources are not consumed, and your application can scale up well.

Once the server machine is set up to support the communication, in order to access data you'll need to declare an instance of the **Database** object in your code, set the **Server** property to the server machine's name or URL, and set the ADO connection string with the **Connect** property. Then you can access data using the methods of the **Database** object.

In summary, these are the features of the data access client components:

- get an ADO recordset via a method call
- retrieve multiple recordsets via a single method call
- execute a SQL statement
- call a stored procedure (with input and/or output parameters)
- retrieve a recordset from an Oracle stored procedure
- use a method that takes a recordset, applies the update (on the remote server machine), and returns a "conflict" recordset consisting of all those records for which an update could not be made
- make your application scale up well, since the component communicates with a server-side ActiveX DLL residing in Microsoft Transaction Server
- avoid the need to install and keep updating ODBC drivers, SQL*Net (for Oracle), or other database connectivity software on every client machine in order for applications to work
- avoid the requirement of a full ADO installation. Only the ADO client components are needed

Because the data access objects client components are so closely tied in with ADO, you should have access to the ADO documentation to be able to fully understand many of the constants and parameters used and referred to in this help.

Database Object

Properties Methods Events

Data is retrieved from a remote data source as disconnected ADO recordsets using the **GetRS** method of the **Database** object. If multiple recordsets are required at once, use the **GetRSMultiple** method to minimize network traffic and optimize your application's performance. Data is updated either using a SQL statement via the **ExecuteSQL** method, by batch update via the **UpdateBatch** method, or by calling a stored procedure with the **ExecuteStoredProc** method. You can obtain schema information with the **GetSchemaInfo** method, which is similar to the ADO **GetSchema** method. If you have stored procedures in an Oracle database that return recordsets, you can retrieve these recordsets using the **GetStoredProcOrclRS** method.

For sophisticated multi-user functionality, you can use the **UpdateBatchConf** method, which returns a "conflict" recordset consisting of all those records for which an update could not be made.

At run time, you can change the **Server** and **Connect** properties to use a different server machine or database.

Parameters Collection

Properties Methods Events

A **Parameters** collection contains all the **Parameter** objects that will be passed into and out of the **ExecuteStoredProc** method of the **Database** object.

Use the **Add** method to create **Parameter** objects with the appropriate property settings and append them to the **Parameters** collection.   Use the **Delete** method to remove **Parameter** objects from the **Parameters** collection if necessary.

Parameter Object

Properties Methods Events

With the collections, methods, and properties of a **Parameter** object, you can do the following:

- Set or return the name of a parameter with the **Name** property.
- Set or return the value of a parameter with the **Value** property.
- Set or return parameter characteristics with the **Attributes** and **Direction**, **Precision**, **NumericScale**, **Size**, and **DataType** properties.

ConflictRecordset Property

See Also Example Applies To

Returns the conflict **Recordset** object that was populated by a call to the **UpdateBatchConf** method. Read-only.

Syntax

object.**ConflictRecordset**

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

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

Remarks

The **ConflictRecordset** cannot be assigned to the **SourceRecordset** property of RemoteDataBar and sent for a subsequent batch update with the **UpdateBatch** or **UpdateBatchConf** methods, since it lacks the field metadata to make it updatable.

Connect Property

See Also Example Applies To

Sets or returns ADO connection string to be used for data.

Syntax

object.**Connect** [= *string*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
string	A string expression that evaluates to a valid ADO connection string.

Remarks

Refer to ADO help for a description of the parts that make up the connection string.

Server Property, Connect Property, FetchMetaData Property Example

This example shows how the **Server**, **Connect**, and **FetchMetaData** properties can be used to prepare for retrieving a read-only **Recordset**.

```
Private Sub Command1_Click()
    Dim Database1 As New DADAO.Database
    Database1.Server = "http://cawebdev"
    Database1.Connect = "Provider=MSDAORA;Data Source=cadev09;User
ID=it_time;Password=it_time"
    Database1.FetchMetaData = False
End Sub
```

Count Property

See Also Example Applies To

Indicates the number of objects in a **Parameters** collection.

Syntax

object.**Count**

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

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

Return Value

Returns a **Long** value.

Remarks

Use the **Count** property to determine how many objects are in a **Parameters** collection.

If you are using Microsoft Visual Basic and want to loop through the members of the collection without checking the **Count** property, you can use the **For Each...Next** statement.

If the **Count** property is zero, there are no objects in the collection.

Database Property

See Also Example Applies To

Returns the **Database** object associated with this control. Read-only.

Syntax

object.**Database**

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

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

Remarks

Use the **Database** object via this property to obtain further ADO recordsets through code, to execute SQL statements, call stored procedures, or retrieve schema information.

FetchMetaData Property

See Also Example Applies To

Sets/returns whether field metadata will be retrieved with the recordset.

Syntax

object.**FetchMetaData** [= *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A boolean expression that determines whether field metadata is fetched from the provider.

Settings

The settings for *value* are:

Setting	Description
True	Field metadata is fetched.
False	Field metadata is not fetched. The recordset will not be updatable.

Remarks

Set to this property to True to allow for an updatable Recordset that can be used in a batch update with the UpdateBatch or UpdateBatchConf methods.

InternetTimeout Property

See Also Example Applies To

Indicates the timeout (in milliseconds) for HTTP transmissions.

Syntax

object.**InternetTimeout** [= *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A **Long** value that determines the number of milliseconds before a method call times out when using HTTP.

Recordset Property

See Also Example Applies To

Returns the ADO **Recordset** object associated with this control. Read-only.

Syntax

object.**Recordset**

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

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

RecordSource Property

See Also Example Applies To

Gets/sets the source of the recordset.

Syntax

object.**RecordSource** [= *string*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
string	A string expression that evaluates to a valid SQL data request.

Remarks

In general, this is a SQL statement (using the dialect of the database server), such as “SELECT * FROM customers”.

Server Property

See Also Example Applies To

Gets or sets the server name and communication protocol.

Syntax

Protocol	**Description**
HTTP	*object*.**Server="http:**//*awebsrvr:port***"**
HTTPS	*object*.**Server="https:**//*awebsrvr:port***"**
DCOM	*object*.**Server="**machinename**"**
In-process	*object*.**Server=""**

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.
awebsrvr or machinename	A **String** that contains a valid Internet or intranet path and server name.

Remarks

The server is the machine where the Data Access Objects server component is located. It need not be the machine where the data source is located. A connection to the server is made using Remote Data Service (refer to ADO Help for more details).

If the DCOM protocol is used, *machinename* is specified without the \\ characters.

SourceRecordset Property

See Also Example Applies To

Sets or returns the underlying ADO **Recordset**.

Syntax

object.**SourceRecordset** [= *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	An object variable containing a **Recordset** object.

Toolbar Property

See Also Example Applies To

Returns the **Toolbar** control contained in this control. Read-only.

Syntax

object.**Toolbar**

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

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

Remarks

Using this property, you may access or modify the toolbar contained in a RemoteDataBar object.

UpdateType Property

See Also Example Applies To

Sets or returns how updates are performed.

Syntax

object.**UpdateType** [= *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A constant or Integer value that specifies an update type, as described in Return Values.

Return Values

Constant	Value	Description
UpdateTypeBatch	0	A batch update is made with a call to the **UpdateBatch** method.
UpdateTypeCurrent	1	Updates are made to the current record when the Save, Add or Delete buttons are pressed.

Attributes Property

See Also Example Applies To

Indicates one or more characteristics of a **Parameter** object. Refer to ADO documentation.

Syntax

object.**Attributes** [= *long*]

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

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

DataType Property

See Also Example Applies To

Indicates the data type of a **Parameter** object. Refer to ADO documentation.

Syntax

object.**DataType** [= *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	An expression that evaluates to a value in the **ADODB.DataTypeEnum** enumeration.

Direction Property

See Also Example Applies To

Indicates whether the **Parameter** represents an input parameter, an output parameter, or both.

Syntax

object.**Direction** [= *long*]

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

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

Remarks

The client-side **Parameter** object supplies a **ParameterDirectionEnum** enumeration that is identical to the **ADODB.ParameterDirectionEnum**. This enumeration is supplied because the ADODB enumeration is not available if only the ADO/RDS client components are installed on the client machine.

Name Property

See Also Example Applies To

Indicates the name of the **Parameter**. Refer to ADO documentation.

Syntax

object.**Name** [= *string*]

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

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

NumericScale Property

See Also Example Applies To

Indicates the scale of numeric values in a **Parameter** object. Refer to ADO documentation.

Syntax

object.**NumericScale** [= *byte*]

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

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

Precision Property

See Also Example Applies To

Indicates the degree of precision for numeric values in a **Parameter** object. Refer to ADO documentation.

Syntax

object.**Precision** [= *byte*]

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

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

Size Property

See Also Example Applies To

Indicates the maximum size, in bytes or characters, of a **Parameter** object. Refer to ADO documentation.

Syntax

object.**Size** [= *long*]

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

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

Value Property

See Also Example Applies To

Indicates the value assigned to a **Parameter** object. Refer to ADO documentation.

Syntax

object.**Value** [= *variant*]

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

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

AboutBox Method

See Also Example Applies To

Displays version information about the control.

Syntax

object.**AboutBox**

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

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

Remarks

Displays the About dialog box for the product. Use this method to display version information about the control.

Add Method

See Also Example Applies To

Creates a new **Parameter** object based on the given arguments and adds it to a **Parameters** collection.

Syntax

object.**Add** (**ByVal** Name **As String, ByVal** Direction **As ParameterDirectionEnum,** [**ByVal** DataType **As ADODB.DataTypeEnum**], [**ByVal** Value **As Variant**])

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
Name	A string expression that evaluates to the name of the new object, which must not be the same as any other object in the collection.
Direction	The type of the **Parameter** object. See the **Direction** property for valid settings.
DataType	The data type of the **Parameter** object. See the **DataType** property for valid settings.
Value	The value for the **Parameter** object.

Remarks

Use the **Add** method to create a new **Parameter** object with the specified name, type, direction, size, and value, and add this to a **Parameters** collection. Any values you pass in the arguments are written to the corresponding **Parameter** properties.

ExecuteSQL Method

See Also Example Applies To

Executes the given SQL statement.

Syntax

object.**ExecuteSQL** (**ByVal** sSQL **As String**, [**ByRef** RecordsAffected **As Long**])

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
sSQL	A string expression that evaluates to a SQL query.
RecordsAffected	A Long variable to which the provider returns the number of records that the operation affected.

ExecuteSQL Method Example

This example shows how the **ExecuteSQL** method can be used to execute a SQL update query.

```
Private Sub Command1_Click()
    Dim RecsAffected As Long
    Database1.ExecuteSQL "UPDATE vendor SET phone = NULL", RecsAffected
End Sub
```

ExecuteStoredProc Method

See Also Example Applies To

Executes a stored procedure.

Syntax

object.**ExecuteStoredProc** (**ByVal** sCmd **As String**, Params **As Parameters**])

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
sCmd	A string expression that evaluates to a stored procedure name.
Params	An object variable containing a **Parameters** collection object.

Remarks

Use **ExecuteStoredProc** rather than **ExecuteStoredProcQ** for better performance in calling a stored procedure. The drawback is that the properties of the parameters must be set correctly.

ExecuteStoredProc, Add, Item, Size, Precision, NumericScale Example

This example shows how the **ExecuteStoredProc** method can be used to execute a stored procedure that takes two input parameters and one output parameter.

```
Private Sub Command1_Click()
    Dim oParams As DADAO.Parameters
    Dim lVendorID As Long

    Set oParams = New DADAO.Parameters
    With oParams
        .Add "iname", adParamInput, adVarChar, txtName
        .Item("iname").Size = 20
        .Add "iowed", adParamInput, adNumeric, 2500
        .Item("iowed").Precision = 6
        .Item("iowed").NumericScale = 2
        .Add "ovendor_id", adParamOutput
    End With
    On Error GoTo SP_Error
    Database1.ExecuteStoredProc "insert_vendor", oParams
    On Error GoTo 0
    lVendorID = oParams.Item("ovendor_id").Value
    Exit Sub

SP_Error:
    MsgBox Err.Description
End Sub
```

ExecuteStoredProcQ Method

See Also Example Applies To

Executes a stored procedure.

Syntax

object.**ExecuteStoredProcQ** (**ByVal** sCmd **As String**, Params **As Parameters**])

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
sCmd	A string expression that evaluates to a stored procedure name.
Params	An object variable containing a **Parameters** collection object.

Remarks

Use **ExecuteStoredProcQ** rather than **ExecuteStoredProc** when performance is not much of an issue, and where you don't want to specify detailed parameter properties. This method causes the parameter information to be retrieved from the provider, which is a potentially resource-intensive operation.

ExecuteStoredProcQ Method Example

This example shows how the **ExecuteStoredProcQ** method can be used to execute a stored procedure that takes two input parameters and one output parameter.   Notice that the **Size**, **Precision** or **NumericScale** properties of the input parameters do not need to be set.

```
Private Sub Command1_Click()
    Dim oParams As DADAO.Parameters
    Dim lVendorID As Long

    Set oParams = New DADAO.Parameters
    With oParams
        .Add "iname", adParamInput, adVarChar, txtName
        .Add "iowed", adParamInput, adNumeric, 2500
        .Add "ovendor_id", adParamOutput
    End With
    On Error GoTo SP_Error
    Database1.ExecuteStoredProcQ "insert_vendor", oParams
    On Error GoTo 0
    lVendorID = oParams.Item("ovendor_id").Value
    Exit Sub

SP_Error:
    MsgBox Err.Description
End Sub
```

GetRS Method

See Also Example Applies To

Retrieves an ADO recordset.

Syntax

object.**GetRS** (**ByVal** sSQL **As String**) **As ADODB.Recordset**

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

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.
sSQL	A string expression that evaluates to a valid SQL statement.

Remarks

Data is retrieved from a remote data source (as specified in the **Connect** property) as a disconnected ADO recordset.

GetRSMultiple Method

See Also Example Applies To

Retrieves multiple ADO recordsets.

Syntax

object.**GetRSMultiple** (**ByVal** sSQL() **As String**) **As ADODB.Recordset()**

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
sSQL	An array of string expressions that evaluate to valid SQL statements.

Return Value

An array of Recordset objects, where each object is a result of the corresponding SQL statement in the sSQL array.

Remarks

This method is extremely useful in situations such as initializing controls in a form, where each control contains data from separate database tables. Instead of calling **GetRS** multiple times to initialize each control, which requires as many round-trip remote procedure calls over a network, use **GetRSMultiple** to minimize network traffic and optimize your application's performance.

GetRSMultiple Method Example

This example shows how the **GetRSMultiple** method may be used to retrieve one recordset containing Customers and one containing Employees.

```
Private Sub Command1_Click()
    Dim asSQL(1) As String
    Dim vaRecordsets As Variant
    Dim rsCustomers As ADODB.Recordset
    Dim rsEmployees As ADODB.Recordset

    asSQL(0) = "SELECT * FROM Customers"
    asSQL(1) = "SELECT * FROM Employees"
    vaRecordsets = Database1.GetRSMultiple(asSQL)
    Set rsCustomers = vaRecordsets(0)
    Set rsEmployees = vaRecordsets(1)
End Sub
```

GetStoredProcOrclRS Method

See Also Example Applies To

Retrieves a recordset from an Oracle stored procedure.

Syntax

object.**GetStoredProcOrclRS** (**ByVal** sCmd **As String**, **ByVal** Params **As Parameters**]) **As ADODB.Recordset**

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
sCmd	A string expression that evaluates to a stored procedure name.
Params	An object variable containing a **Recordset** object.

Remarks

The Parameter objects contained in the Params argument may be input parameters only.

See Microsoft Knowledge Base article Q176086. Note that the Microsoft ODBC for Oracle driver must be used to enable this feature.

GetStoredProcOrclRS Method Example

This example shows how the **GetStoredProcOrclRS** method can be used to retrieve a **Recordset** from a stored procedure in an Oracle database that takes two (input) parameters.

```
Private Sub Command1_Click()
    Dim Database1 As New DADAO.Database
    Dim Params As DADAO.Parameters
    Dim sCmd As String

    Database1.ServerName = "cawt26"
    Database1.ConnectString = "DRIVER={Microsoft ODBC for
Oracle};UID=co_plan;PWD=co_plan;CONNECTSTRING=cadev09"

    Set Params = New DADAO.Parameters
    With Params
        .Add "iprod_group_id", adParamInput, adNumeric, 1
        .Add "iscenario_id", adParamInput, adNumeric, 1
    End With

    sCmd = "{call gen_projections.proj_by_month(?,?," & _
        "{resultset 60, omonth, omole_units, omole_sales})}"
    On Error GoTo SP_Error
    Set rs = Database1.GetStoredProcOrclRS(sCmd, Params)
    Set DataGrid1.DataSource = rs
    Exit Sub

SP_Error:
    MsgBox Err.Description
End Sub
```

GetSchemaInfo Method

See Also Example Applies To

Gets database schema information from the provider.

Syntax

object.**GetSchemaInfo** (**ByVal** Schema **As ADODB.SchemaEnum**, [**ByVal** Restrictions], **ByVal** SchemaID)

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

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.
Schema	
Restrictions	
SchemaID	

Remarks

The **GetSchemaInfo** method returns information about the data source, such as information about the tables on the server and the columns in the tables.

See ADO help for details.

Item Method

See Also Example Applies To

Returns a specific member of a collection by name or ordinal number.

Syntax

object.**Item** (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	A **Variant** that evaluates either to the name or to the ordinal number of an object in a collection.

Return Value

Returns an object reference.

Remarks

Use the **Item** method to return a specific object in a collection. If the method cannot find an object in the collection corresponding to the ***Index*** argument, an error occurs.

The **Item** method is the default method for the **Parameters** collection; therefore, the following syntax forms are interchangeable:

collection.**Item (*Index*)**
collection **(*Index*)**

RefreshData Method

See Also Example Applies To

Refreshes underlying recordset from provider.

Syntax

object.**RefreshData**

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

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

Remarks

This method is particularly useful in multi-user situations, where conflicting updates may occur; records may be deleted or modified by other users. After a call to **UpdateBatchConf**, if there are conflicts then the user may be presented with the most current data via a call to **RefreshData**.

Remove Method

See Also Example Applies To

Removes a **Parameter** object from a **Parameters** 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	A string expression representing the name of the object you want to remove, or the object's ordinal position (index) in the collection.

Remarks

Using the **Remove** method on a collection lets you remove one of the objects in the collection. You must use the **Parameter** object's **Name** property or its collection index when calling the **Remove** method—an object variable is not a valid argument.

UpdateBatch Method (RemoteDataBar)

See Also Example Applies To

Sends changed records to the server for a batch update.

Syntax

object.**UpdateBatch** ([**ByVal** AffectRecords **As ADODB.AffectEnum** = **adAffectAll**])

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
AffectRecords	An **ADODB.AffectEnum** value that determines how many records the **UpdateBatch** method will affect. See ADO help.

UpdateBatch Method (Database)

See Also Example Applies To

Sends changed records to the server for a batch update.

Syntax

object. **UpdateBatch** (**ByRef** rs **As ADODB.Recordset**, [**ByVal** AffectRecords **As ADODB.AffectEnum** = **adAffectAll**])

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
rs	An object variable containing a **Recordset** object.
AffectRecords	An **ADODB.AffectEnum** value that determines how many records the **UpdateBatch** method will affect. See ADO help.

UpdateBatchConf Method (RemoteDataBar)

See Also Example Applies To

Updates changed records and allows conflicts to be obtained.

Syntax

object.**UpdateBatchConf** ([**ByVal** AffectRecords **As ADODB.AffectEnum** = **adAffectAll**])

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

Part	Description

object	An object expression that evaluates to an object in the Applies To list.
AffectRecords	An **ADODB.AffectEnum** value that determines how many records the **UpdateBatch** method will affect. See ADO help.

Remarks

For **RemoteDataBar**, conflicts can be obtained with the **ConflictRecordset** property.

If there were no conflicts, call **UpdateBatch** (on disconnected recordset) to set UnderlyingValue to Value for all records.

UpdateBatchConf Method (Database)

See Also Example Applies To

Updates changed records and allows conflicts to be obtained.

Syntax

object.**UpdateBatchConf** (**ByRef** rs **As ADODB.Recordset**, **ByRef** rsConf **As ADODB.Recordset**, [**ByVal** AffectRecords **As ADODB.AffectEnum** = **adAffectAll**])

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
AffectRecords	An **ADODB.AffectEnum** value that determines how many records the **UpdateBatch** method will affect. See ADO help.

Remarks

Use the **rsConf** argument to present information to the user about the conflicts that occurred.

UpdateBatchConf Method, ConflictRecordset Property, RefreshData Method Example

This example shows how the **ConflictRecordset** property may be used in conjunction with the **UpdateBatchConf** method. **UpdateBatchConf** performs the batch update and allows detection of conflicts. If any conflicts occurred, the **DataGrid** is populated with the conflicting records, and the **RemoteDataBar** object is refreshed with the current data.

```
Private Sub Command1_Click()
    Me.MousePointer = vbHourglass
    DARemDataBar1.UpdateBatchConf
    If DARemDataBar1.ConflictRecordset Is Nothing Then
        ' If there were no conflicts, call UpdateBatch so that _
          changed records do not get re-submitted (UpdateBatch causes _
          UnderlyingValue to be set to Value for all records).
        DARemDataBar1.Recordset.UpdateBatch
    Else
        DARemDataBar1.RefreshData
        Set DataGrid1.DataSource = DARemDataBar1.ConflictRecordset
    End If
    Me.MousePointer = vbDefault
End Sub
```

AddRecord Method

See Also Example Applies To

Appends a new record to the recordset.

Syntax

object.**AddRecord**

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

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

Remarks

Use the **AddRecord** method to add a new record as a result of a user interface action, for example, when an “Add New” button is pressed.   This method is most commonly used when you are using your own recordset navigation buttons or toolbar instead of the built-in toolbar.

If the **UpdateType** property is set to **UpdateTypeBatch**, then the record will only be added to the local recordset, and will only be added on the server when the **UpdateBatch** method is called.

DeleteRecord Method

See Also Example Applies To

Deletes the current record from the recordset.

Syntax

object.**DeleteRecord**

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

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

Remarks

Use the **DeleteRecord** method to delete the current record as a result of a user interface action, for example, when a "Delete" button is pressed. This method is most commonly used when you are using your own recordset navigation buttons or toolbar instead of the built-in toolbar.

If the **UpdateType** property is set to **UpdateTypeBatch**, then the record will only be deleted from the local recordset, and will only be deleted on the server when the **UpdateBatch** method is called.

FirstRecord Method

See Also Example Applies To

Moves to the first record in the recordset.

Syntax

object.**FirstRecord**

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

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

Remarks

Use the **FirstRecord** method to move to the first record as a result of a user interface action, for example, when a “First Record” button is pressed. This method is most commonly used when you are using your own recordset navigation buttons or toolbar instead of the built-in toolbar.

LastRecord Method

See Also		Example		Applies To

Moves to the last record in the recordset.

Syntax

object.**LastRecord**

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

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

Remarks

Use the **LastRecord** method to move to the last record as a result of a user interface action, for example, when a “Last Record” button is pressed.   This method is most commonly used when you are using your own recordset navigation buttons or toolbar instead of the built-in toolbar.

NextRecord Method

See Also Example Applies To

Moves to the next record in the recordset.

Syntax

object.**NextRecord**

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

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

Remarks

Use the **NextRecord** method to move to the next record as a result of a user interface action, for example, when a “Next Record” button is pressed. This method is most commonly used when you are using your own recordset navigation buttons or toolbar instead of the built-in toolbar.

PreviousRecord Method

See Also   Example   Applies To

Moves to the previous record in the recordset.

Syntax

object.**PreviousRecord**

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

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

Remarks

Use the **PreviousRecord** method to move to the previous record as a result of a user interface action, for example, when a “Previous Record” button is pressed.   This method is most commonly used when you are using your own recordset navigation buttons or toolbar instead of the built-in toolbar.

SaveRecord Method

See Also　　Example　　Applies To

Saves changes to the current record.

Syntax

object.**SaveRecord**

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

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

Remarks

Use the **SaveRecord** method to save the current record as a result of a user interface action, for example, when a “Save” button is pressed.   This method is most commonly used when you are using your own recordset navigation buttons or toolbar instead of the built-in toolbar.

If the **UpdateType** property is set to **UpdateTypeBatch**, then the record will only be updated in the local recordset, and will only be updated on the server when the **UpdateBatch** method is called.

UndoRecord Method

See Also Example Applies To

Restores fields in the current record to their original values.

Syntax

object.**UndoRecord**

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

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

Remarks

Use the **UndoRecord** method to undo editing changes to a record before the “Save” button is pressed. This method is most commonly used when you are using your own recordset navigation buttons or toolbar instead of the built-in toolbar.

BeforeDelete Event

See Also Example Applies To

Fired before the current record is deleted.

Syntax

Private Sub *object*_**BeforeDelete** ([*index* **As Integer**,] rs **As ADODB.Recordset**, **ByVal** db As **Database**, *adStatus* **As ADODB.EventStatusEnum**)

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	An integer that identifies a control if it is in a control array.
rs	An object variable containing a **Recordset** object.
db	An object variable containing a **Database** object.
adStatus	An **ADODB.EventStatusEnum** status value. Refer to ADO Help.

BeforeInsert Event

See Also		Example		Applies To

Fired before the current record is inserted.

Syntax

Private Sub *object*_**BeforeInsert** ([*index* **As Integer**,] rs **As ADODB.Recordset**, **ByVal** db As **Database**, *adStatus* **As ADODB.EventStatusEnum**)

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

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.
index	An integer that identifies a control if it is in a control array.
rs	An object variable containing a **Recordset** object.
db	An object variable containing a **Database** object.
adStatus	An **ADODB.EventStatusEnum** status value. Refer to ADO Help.

BeforeInsert Event, GetRS Method Example

This example shows how an Oracle sequence can be used to fill in the value of a primary key field of a record that is about to be inserted into the local recordset.   This technique mimics the Microsoft Access “AutoNumber” functionality.

```
Private Sub DARemDataBar1_BeforeInsert(rs As ADODB.Recordset, _
        ByVal db As DADAO.Database, adStatus As ADODB.EventStatusEnum)
    Dim sSQL As String
    Dim rsTmp As Recordset

    sSQL = "SELECT vendor_seq.nextval FROM DUAL"
    Set rsTmp = db.GetRS(sSQL)
    rs("vendor_id") = rsTmp(0).Value
End Sub
```

BeforeUndo Event

See Also Example Applies To

Fired before changes are undone to the current record.

Syntax

Private Sub *object*_**BeforeUndo** ([*index* **As Integer**,] rs **As ADODB.Recordset**, **ByVal** db As **Database**, *adStatus* **As ADODB.EventStatusEnum**)

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	An integer that identifies a control if it is in a control array.
rs	An object variable containing a **Recordset** object.
db	An object variable containing a **Database** object.
adStatus	An **ADODB.EventStatusEnum** status value. Refer to ADO Help.

BeforeUpdate Event

See Also   Example   Applies To

Fired before the current record is updated.

Syntax

Private Sub *object*_**BeforeUpdate** ([*index* **As Integer**,] rs **As ADODB.Recordset**, **ByVal** db As **Database**, *adStatus* **As ADODB.EventStatusEnum**)

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	An integer that identifies a control if it is in a control array.
rs	An object variable containing a **Recordset** object.
db	An object variable containing a **Database** object.
adStatus	An **ADODB.EventStatusEnum** status value. Refer to ADO Help.