Testing candidate 4.00.46007/21/95 10:41 AM

Remote Data Objects Overview

See Also

Remote Data Objects implement a set of objects to deal with the special requirements of remote data access. RDO implements a thin code layer over the ODBC API and driver manager that establishes connections, creates result sets and cursors, and executes complex procedures using minimal workstation resources.

With RDO and the **RemoteData** control, your applications can access ODBC data sources without using a local query processor. This can mean significantly higher performance and more flexibility when accessing remote database engines. Although you can access any ODBC data source with RDO and the **RemoteData** control, these features are designed to take advantage of database servers, like Microsoft SQL Server and Oracle, that use sophisticated query engines.

By using RDO, you can create simple cursor-less result sets, or more complex cursors. You can also run queries that return any number of result sets, or execute stored procedures that return result sets with or without output parameters and return values. You can limit the number of rows returned and monitor all of the messages and errors generated by the remote data source without compromising the executing query. RDO also permits either synchronous or asynchronous operation so your application doesn't need to be blocked while lengthy queries are executed.

Note RDO and the **RemoteData** control are features of the Visual Basic, Enterprise Edition. You cannot develop code or use the RDO object library or **RemoteData** control in the Professional or Standard Editions of Visual Basic.

Converting DAO to RDO

Basically, you can use the Remote Data Objects similarly to the way you use the Microsoft Jet database engine Data Access Objects (DAO). With RDO, you can submit queries, create a result set or cursor, and process the results from the query using database-independent object-oriented code.

The following table lists RDO objects and their equivalent DAO objects. In some cases, these two object models are similar in function and operation. However, there are significant differences that are documented in the Help topics.

DAO object or term	RDO object or term	Notes
DBEngine	**rdoEngine**	Base object
Workspace	**rdoEnvironment**	Object and collection
Database	**rdoConnection**	Object and collection
Recordset	**rdoResultset**	Object and collection
Dynaset-type **Recordset**	Keyset-type **rdoResultset**	Object and collection
Snapshot-type **Recordset**	Static-type **rdoResultset**	Object and collection
Not implemented	Dynamic-type **rdoResultset**	No DAO equivalent
Table-type **Recordset**	Not implemented	No RDO equivalent
Not implemented	Forward-only–type **rdoResultset**	No Jet equivalent
forward-only-scrolling snapshot-type **Recordset**	Not implemented	No RDO equivalent
Error	**rdoError**	Object and collection
Field	**rdoColumn**	Object and collection
QueryDef	**rdoPreparedStatement**	Object and collection
TableDef	**rdoTable**	Object and collection

Parameter	**rdoParameter**	Object and collection
record	row	Term
field	column	Term

Jet's table-type **Recordset** object permits direct access to an entire table using a selected ISAM index. Although you could simulate this with RDO, it would require the creation of a cursor that spans the entire table, which is generally impractical. Jet's DAO also provides the ability to create a forward-only-scrolling snapshot-type **Recordset**. This option is similar to the forward-only result set except that the forward-only **rdoResultset** can be updatable.

See Also

Choosing a Cursor Type

Using the **RemoteData** Control

Using the RemoteData Control

See Also

This topic provides additional details on the **RemoteData** control.

The **RemoteData** control performs a number of automatic operations, including:

- Once the form containing the **RemoteData** control is loaded, if sufficient properties have been set at design time, Visual Basic uses the **RemoteData** control to establish a connection to the ODBC data source. This creates an **rdoConnection** object by using the **DataSourceName**, **UserName**, **Password**, **Options**, **Type**, and **Connect** properties. See the following table for the minimum properties required for automatic initialization and how these properties are used.
- If insufficient information is provided in the **RemoteData** control properties, the ODBC driver manager exposes a dialog to gather missing parameters. If the connection is established, the **RemoteData** control sets or resets the **Environment**, **Connection**, **DataSourceName**, **Transactions**, and **Connect** properties based on the values used to establish the connection.
- Once the connection is established, the **RemoteData** control runs a query against the data source using the **SQL, CursorDriver**, **Options**, **LockType**, **ErrorThreshold**, and **ResultsetType** properties. This creates an **rdoResultset** object and sets the **Resultset**, **ResultsetType**, and **Updatable** properties. By default, a read-only, forward-only **rdoResultset** is created if the **ResultsetType** is not specified or is not supported by the driver.
- The **StillExecuting** property is set to **True** while the **rdoResultset** is being created. If you choose to cancel the query, and the **rdAsyncEnable** option is set, you can use the **Cancel** method against the **rdoResultset** to terminate processing of the query.
- Once the first row of the **rdoResultset** is available, the **StillExecuting** property is set to **False** and the **RemoteData** control passes column data to each bound control requesting data. The **rdoResultset**.**RowCount** property is set to a non-zero value if any rows resulted from the query. If no data is returned by the **rdoResultset**, the **RemoteData** control's behavior is determined by the **EOFAction** property.

RDO Support

The **RemoteData** control also exposes the component objects it uses or creates so that your code can use these objects with appropriate RDO methods and properties.

RemoteData property	Exposes this RDO object	Access
Environment	**rdoEnvironment**	**Read-Only**
Connection	**rdoConnection**	**Read-Only**
Connect	*Connect* **strings**	**Read/Write**
Resultset	**rdoResultset**	**Read/Write**

You can also create an **rdoResultset** independently of the **RemoteData** control and set the **Resultset** property to this new object. The **RemoteData** control assumes the **rdoConnection** and other attributes of the assigned **rdoResultset** object.

Validation

Use the Validate event and the **DataChanged** property to perform last-minute checks on the rows about to be written to the database. The Validate event is triggered before each reposition of the current row pointer. If data changes in any bound control, the Validate event is triggered, and if not canceled by the *action* argument, the data source is updated. See the Validate event for more information.

BOF/EOF Handling

The **RemoteData** control can also manage what happens when you encounter an **rdoResultset** with no rows. By changing the **EOFAction** property, you can program the **RemoteData** control to enter **AddNew** mode automatically.

You can program the **RemoteData** control to automatically snap to the top or bottom of its parent form by using the **Align** property.   In either case, the **RemoteData** control is resized horizontally to fill the width of its parent form whenever the parent form is resized. This property allows a **RemoteData** control to be placed on an MDI form without requiring an enclosing **Picture** control.

See Also

Remote Data Objects Overview
Repositioning the Current Row Pointer
Running Asynchronous Queries
Using **RemoteData** Control Properties
Understanding **rdoResultset** Objects
Using Bound Controls with the **RemoteData** Control
Using ODBC Handles

Creating Parameter Queries

See Also Example

When you submit a query to a remote query processor for execution, you must often modify the query to refocus the scope of the result set. For example, if you need to create a result set that contains all of the dentists in a particular town, you can use the following code:

```
"SELECT Name, Specialty, Phone FROM Dentists WHERE Town = 'Detroit' "
```

The desired town is hard-coded in the query, so you must write code to change the town name each time the query is executed.

You can handle this problem by:

- Concatenating the SQL parameters that change into the SQL query.
- Coding the query so that RDO and the ODBC driver manager can integrate the parameters as required.

Concatenating the SQL Parameters

This technique uses a base query that is created each time it is executed. The desired parameter(s) are appended into the SQL string in the appropriate place(s). For example:

```
Dim MySQL As String
MySQL = "SELECT Name, Specialty, Phone FROM Dentists WHERE Town = '"
MySQL = MySQL & SelectedTown$ & "' "
```

The resulting string is passed to the **OpenResultset** method for execution. This approach is simple to use, but does not lend itself to parameters that must be returned as well as passed. It also requires that the remote query processor re-compile the query each time it is executed, which can impact performance on heavily used systems. You must also remember to bracket the parameter in quotes and deal with the eccentricities of embedded quotes and formatting.

Coding the Query to Integrate the Parameters

By using the ODBC escape-code placeholders, you can specify four types of parameters: input, output, input/output, and return values. When working with certain types of stored procedures, this technique is the only suitable method to pass and return parameters. For these situations, use ODBC SQL syntax to create your query, and a question mark "?" as a placeholder for your return value and each parameter. For example, to code the Dentist example:

```
MySQL = "SELECT Name, Specialty, Phone FROM Dentists WHERE Town = ?"
```

This string is used as an argument when you create an **rdoPreparedStatement** object using the **rdoConnection** object's **CreatePreparedStatement** method. Before you execute the query, you must set the parameter values using the **rdoPreparedStatement** object's **rdoParameters** collection. For example:

```
Dim Ps As rdoPreparedStatement
' cn is an open rdoConnection.
Set Ps = cn.CreatePreparedStatement("PSDentist", MySQL)
Ps.rdoParameters(0) = SelectedTown$
```

To execute the query, use the **OpenResultset** method against the **rdoPreparedStatement**:

```
Dim Rs As rdoResultset
Set Rs = Ps.OpenResultset(rdOpenKeyset)
```

To change the parameter(s) for subsequent queries, simply change the **rdoParameter** for each parameter in the query, and use the **Requery** method to run the query again:

```
Ps.rdoParameters(0) = SelectedTown$' Assuming this has changed.
Rs.Requery
```

Return Codes and Output Parameters

If your query returns information in the form of output parameters or a return value, you must set the **Direction** property of the non-input parameters before you execute the query. Since return values and output parameters are returned by stored procedures, you must use a slightly different syntax to execute the query. In this case, you must use the ODBC *call* operator to help the ODBC interface parse and process the query correctly. For example, to execute a procedure that has a return value, an input parameter, and an output parameter, you need to create an SQL statement that contains the name of the stored procedure:

```
MySQL = "{ ? = call My_StoredProc (?, ?) }"
```

Once the procedure is defined, incorporate it into an **rdoPreparedStatement** and set the **Direction** property based on the type of parameter:

```
Set Ps = cn.CreatePreparedStatement("PSStoredProc", MySQL
Ps(0).Direction = rdParamReturnValue
Ps(1).Direction = rdParamInput
Ps(2).Direction = rdParamOutput
```

At this point you are ready to execute the procedure. Use the **OpenResultset** method or the **Execute** method (for action queries) against the **rdoPreparedStatement** to execute the query:

```
Ps(1) = "My Input value"
Set Rs = rdoPreparedStatement.OpenResultset(rdOpenKeyset)
```

To retrieve the values returned by the return value and the output parameters, reference the **rdoParameters** collection:

```
MyRetVal = Ps(0)
MyOutputVal = Ps(2)
```

As with the simpler form of **rdoPreparedStatement**, you can use the **Requery** or **Execute** method (for action queries) to re-execute the query after having changed one or more parameters.

See Also

Direction Property

OpenResultset Method

rdoParameter Object, **rdoParameters** Collection

rdoPreparedStatement Object, **rdoPreparedStatements** Collection

rdoResultset Object, **rdoResultsets** Collection

Requery Method

Creating Parameter Queries Example

The following example accepts a name and city from two **TextBox** controls and executes a function that runs a query that passes these two values as arguments. The function returns a value from a stored procedure and creates an **rdoResultset** containing all matching rows.

```
Option Explicit
Public ps As rdoPreparedStatement
Public cn As rdoConnection
Public rs As rdoResultset
Private Sub OpenRDOConnection()
   Set cn = rdoEnvironments(0).OpenConnection("sequel", _
    , , "uid=;pwd=;database=pubs;")
   Set ps = cn.CreatePreparedStatement("GetAuthor", "")
   ps.LogMessages = "C:\vb.tst\sqllog.out"
   ps.SQL = "Select * from authors where au_lname = ? and state = ?"
End Sub

Private Function SubmitQuery() As Integer
Dim i As Integer
ps.rdoParameters(0) = NameWanted.Text    ' Set second input parameter.
ps.rdoParameters(1) = StateWanted.Text   ' Set first input parameter.

' Create result set based on parameters.
Set rs = ps.OpenResultset(rdOpenKeyset)

' Return the return value from stored procedure.
SubmitQuery = ps.rdoParameters(0)

If rs.EOF Then
   SubmitQuery = -1      ' If no data is returned, indicate by -1.
Else
   List1.Clear
   While Not rs.EOF
      List1.AddItem rs(0) & " - " & rs(1) & ", " & rs(2)
         rs.MoveNext
   Wend
End If
End Function

Private Sub Command1_Click()
   Dim Result%
   Result = SubmitQuery()
End Sub

Private Sub Form_Load()
   OpenRDOConnection
End Sub
```

Understanding Cursors

See Also

Applications have different needs in their ability to sense changes in the underlying tables of a result set. For example, when balancing financial data, an accountant needs data that appears static; it is impossible to balance books when the data is continually changing. When selling concert tickets, a clerk needs up-to-the-minute, or dynamic, data on which tickets are still available. Various cursor models are designed to meet these needs, each of which requires different sensitivities to changes in the underlying tables of the result set.

Forward-Only "Cursors"

The fastest way to access data from a remote data source is through use of the forward-only result set − this type of **rdoResultset** is not really a cursor. In this case, the **rdoResultset** is returned without the ability to move the current row back to previously-fetched rows.

For forward-only result sets, use **rdOpenForwardOnly** as the type in the **OpenResultset** method.

Static Cursors

At one extreme are *static* cursors, to which the data in the underlying tables appears to be static. The membership, order, and values in the result set used by a static cursor are generally fixed when the cursor is opened. Rows updated, deleted, or inserted by other users (including other cursors in the same application) are not detected by the cursor until it is closed and then reopened.

Static cursors are commonly implemented by taking a snapshot of the data or locking the result set. Note that in the former case, the cursor diverges from the underlying tables as other users make changes; in the latter case, other users are prohibited from changing the data.

For static cursors, use **rdOpenStatic** as the type in the **OpenResultset** method or in the **ResultsetType** property of the **RemoteData** control.

Dynamic Cursors

At the other extreme are dynamic cursors, where the membership, order, and values in the result set are ever-changing. Rows updated, deleted, or inserted by all users (your cursor, other cursors in the same application, and other applications) are detected by the cursor when data is accessed. Although ideal for many situations, dynamic cursors are difficult and expensive to implement.

For dynamic cursors, use **rdOpenDynamic** as the type in the **OpenResultset** method.

Keyset Cursors

Between static and dynamic cursors are keyset-driven cursors, which have some of the attributes of each. Like static cursors, the membership and ordering of the result set of a keyset-driven cursor is generally fixed when the cursor is opened. Like dynamic cursors, changes to the values in the underlying result set are visible to the cursor when data is next fetched.

When a keyset-driven cursor is opened, the driver saves the keys for the entire result set, thus fixing the membership and order of the result set. As the cursor scrolls through the result set, the driver uses the keys in this *keyset* to retrieve the current data values for each row in the rowset. Because data values are retrieved only when the cursor scrolls to a given row, updates made to that row by other users (including other cursors in the same application) are visible.

If the cursor scrolls to a row of data that has been deleted by other users (including other cursors in the same application), the row appears as a *hole* in the result set, since the key is still in the keyset but the row is no longer in the result set. Updating the key values in a row is treated as if the existing row is deleted and a new row is inserted; therefore, rows of data for which the key values have been changed also appear as holes. When the driver encounters a hole in the result set, it returns a trappable error indicating that the row has been deleted.

Rows of data inserted into the result set by other users (including other cursors in the same application) after the cursor was opened are not visible to the cursor, since the keys for those rows are not in the keyset.

In some cases, the cursor can detect rows it has deleted or inserted. Because updating key values in a keyset-driven cursor is treated as if the existing row is deleted and a new row is inserted, keyset-driven cursors can always detect rows they have updated.

For keyset cursors, use **rdOpenKeyset** as the type in the **OpenResultset** method or in the **ResultsetType** property of the **RemoteData** control.

Mixed (Keyset/Dynamic) Cursors

If a result set is large, it may be impractical for the driver to save the keys for the entire result set. Instead, the application can use a *mixed* cursor. In a mixed cursor, the keyset is smaller than the result set, but larger than the rowset.

Within the boundaries of the keyset, a mixed cursor is keyset-driven−that is, the driver uses keys to retrieve the current data values for each row in the rowset. When a mixed cursor scrolls beyond the boundaries of the keyset, it becomes dynamic

−that is, the driver simply retrieves the next *rowset size* rows of data. The driver then constructs a new keyset, which contains the new rowset.

For example, assume a result set has 1000 rows and uses a mixed cursor with a keyset size of 100 and a rowset size of 10. When the cursor is opened, the driver (depending on the implementation) saves keys for the first 100 rows and retrieves data for the first 10 rows. If another user deletes row 11 and the cursor then scrolls to row 11, the cursor will detect a hole in the result set; the key for row 11 is in the keyset but the data is no longer in the result set. This is the same behavior as a keyset-driven cursor. However, if another user deletes row 101 and the cursor then scrolls to row 101, the cursor will not detect a hole; the key for row 101 is not in the keyset. Instead, the cursor will retrieve the data for the row that was originally row 102. This is the same behavior as a dynamic cursor.

For keyset cursors, use **rdOpenKeyset** as the type in the **OpenResultset** method and set the keyset size smaller than the expected result set row count, but larger than the rowset size.

See Also

Choosing a Cursor Type
CursorDriver Property
OpenResultset Method
rdoDefaultCursorDriver Property
rdoResultset Object, **rdoResultsets** Collection
Repositioning the Current Row Pointer
Using the **RemoteData** Control

Understanding rdoResultset Objects

See Also

Your application can create as many **rdoResultset** object variables as needed. An **rdoResultset** can refer to one or more tables based on SQL queries that join data from base tables. **rdoResultset** objects cannot refer to other **rdoResultset** objects or **rdoPreparedStatement** objects.

You can have more than one **rdoResultset** object variable that refers to the same set of rows. Also, keep in mind that object variables created with the **Dim** statement in the Declarations section of a form exist only while the form is open, and are closed when the form is unloaded. In the same way, object variables created in a procedure exist only until the procedure has finished. Explicitly complete any pending transactions or edits, and close all **rdoResultset** and **rdoConnection** objects before exiting or completing any procedure containing declarations for these remote data objects.

An **rdoResultset** may not be updatable even if you request an updatable **rdoResultset**. If the underlying database, table, or column isn't updatable, or if your user does not have update permission, all or portions of your **rdoResultset** may be read-only. Examine the **rdoConnection**, **rdoResultset**, and **rdoColumn** objects' **Updatable** property to determine if your code can change the rows.

Note If you use variables to represent an **rdoResultset** object and the **rdoConnection** object that contains the **rdoResultset** object, make sure the variables have the same scope or lifetime. For example, if you declare a global variable that represents an **rdoResultset**, make sure the variable that represents the database containing the rowset is also global or is declared in a **Sub** or **Function** procedure using the **Static** keyword.

See Also

rdoResultset Object, **rdoResultsets** Collection

Understanding Cursors

Using Bound Controls with the RemoteData Control

See Also

The **RemoteData** control can be used with any data-aware bound control supplied with Visual Basic or from third parties.

Generally, bound controls expose the data values from the current row of the **rdoResultset** created by the **RemoteData** control. However, the **DBList**, **DBCombo**, and **DBGrid** controls are all capable of managing sets of rows when they are bound to a **RemoteData** control. All of these controls permit several rows to be displayed or manipulated as a set.

The **CheckBox**, **TextBox**, **Label**, **Picture**, **Image**, **ListBox**, and **ComboBox** controls are also data-aware. Each of these controls can be bound to a single column of an **rdoResultset** managed by the **RemoteData** control. Additional data-aware controls like the **RichText**, **MaskedEdit**, and **3DCheckBox** controls are also available in the Professional and Enterprise Editions of Visual Basic.

Make sure that each bound control's **DataField** property corresponds to the **Name** property of an **rdoColumn** object returned by an **rdoResultset**. If the **DataField** property does not match the **Name** property of one of the **rdoColumn** objects in an **rdoResultset**, a trappable error results.

Note Help files and other documentation for bound controls might not refer to the correct **RemoteData** control properties when reporting errors or when discussing their properties or use. For example, some bound control documentation mentions the **RecordSource** property. When using the **RemoteData** control, instead of the **Data** control, the **SQL** property is used in roughly the same fashion as the **RecordSource** property.

See Also

rdoResultset Object, **rdoResultsets** Collection
RemoteData Control
Resultset Property
SQL Property
Using the **RemoteData** Control

Repositioning the Current Row Pointer

See Also

An **rdoResultset** object can have any number of valid rows, but only one of those rows can be current at any one time. Most RDO operations are made against, or in relation to the current row. For example, when you use the **Edit** method, you activate the current row for editing. When you use the **MoveNext** or **MovePrevious** method, the current row is repositioned relative to the current row.

The current row might not point to a valid row, or may be indeterminate. For example, if you use the **Delete** method, the current row is no longer valid and is considered to be in an indeterminate state. An **rdoResultset** can be created that has no rows−in this case there is no current row.

Repositioning the Current Row Using RDO

You can use the following *Move* methods against an open **rdoResultset** object:

- **MoveNext** moves the current row pointer one row toward the end of the **rdoResultset**.
- **MovePrevious** moves the current row pointer one row toward the beginning of the **rdoResultset**.
- **MoveLast** moves the current row pointer to the last row (the end) of the **rdoResultset**.
- **MoveFirst** moves the current row pointer to the first row (the beginning) of the **rdoResultset**.
- **Move** moves the current row pointer '*n*' rows forward or backward relative to the current row or a bookmark.

You can also reposition the current row by using one of the movement properties against an open **rdoResultset** object:

- **Bookmark** moves the current row pointer to a specific row in the **rdoResultset** based on a saved bookmark value.
- **AbsolutePosition** moves the current row pointer to the '*n*th' row in the **rdoResultset**.
- **PercentPosition** moves the current row pointer to a specified percentage value based on the number of rows in the **rdoResultset**. For example, if you set **PercentPosition** to 10 and there are 200 rows in the result set, the current row is positioned on the 20th row.
- **LastModified** returns the bookmark of the row which was most recently modified. If you set the **Bookmark** property to this value, the current row pointer is repositioned to that row.

Note If you create a forward-only result set, only the **MoveNext** or **MoveLast** methods or **Move** with a positive value can be used.

Repositioning the Current Row Using the RemoteData Control

You can reposition the current row pointer using the mouse and the **RemoteData** control, or by using any of the *Move* methods against its **Resultset** property. The following details examine what happens when the current row pointer is changed.

- Before each reposition, the bound controls are queried for new data for the current row. The Validate event is triggered, and if not canceled by the *action* argument, the data source is updated if any data in the bound control has changed.
- After the **RemoteData** control positions to a new row in the data source, column data is passed to the bound controls and the Reposition event is triggered.
- Once either end of the **rdoResultset** is reached, the **RemoteData** control's behavior is determined by the **EOFAction** and **BOFAction** properties.
- You can use the **MoreResults** method against the **rdoResultset** to complete processing of the current result set and determine if additional result sets are available. If **MoreResults** returns **True**, the process of handling the **rdoResultset** is restarted

− just as if a new query has been executed.   The previous result set is no longer available.

See Also

rdoResultset Object, **rdoResultsets** Collection
RemoteData Control
Resultset Property
SQL Property
Using the **RemoteData** Control

Using ODBC Handles

See Also

RDO and the **RemoteData** control provide access to the underlying ODBC handles used by the ODBC driver manager and the ODBC data source driver to create the data objects and manage the environment and connection. These handles are created and released automatically and are accessible by using the appropriate property. You should not save these handles in program variables as they are subject to change without notice.

The following table shows the ODBC handles that are created by RDO.

RDO property	Handle created by the ODBC API
rdoEnvironment.hEnv	**SQLAllocEnv**
rdoConnection.hDbc	**SQLAllocConnect, SQLDriverConnect**
rdoResultset.hStmt	**SQLAllocStmt**
rdoPreparedStatement.hStmt	**SQLAllocStmt**

Warning If you close connections or deallocate any of these handles using ODBC API code, the **RemoteData** control and RDO might behave unpredictably.

See Also

hDbc Property
hEnv Property
hStmt Property

Running Asynchronous Queries

See Also

If you need to execute a query that is expected to run for an extended period of time, you can use the **rdAsyncEnable** option.   In this case, RDO begins execution of the query, but returns control to your application immediately—before the query is completed or the result set is created.   This is called asynchronous operation, and can be used with queries that return rows, or with action queries.

RDO Asynchronous Operation

To start an asynchronous query using RDO, set the *option* argument in the **OpenResultset** or **Execute** methods to **rdAsyncEnable**.   Once the query is started, control returns to your application immediately.   You cannot use the **rdoResultset** until the query is complete as indicated by the **StillExecuting** property.   Once **StillExecuting** changes to **False**, the **rdoResultset** can be accessed, or you can examine the **RowsAffected** property.

RDO periodically checks to see if the query is complete.   You can set the period of time between checks by using the **AsyncCheckInterval** property.

If you choose to stop execution of a query, you can use the **MoreResults** method, which flushes the current **rdoResultset** and prepares the next result set.   This is especially useful in cases where the query returns more than one result set.   If **MoreResults** returns **False**, there are no additional result sets to process.

To flush an entire query including all subsequent result sets, use the **Cancel** method against the result set.   This stops execution of the entire query batch—not just the currently running query.

If you do not request asynchronous operation, no other Visual Basic operations or events can occur until the first data row of the **rdoResultset** is fetched—your application is essentially blocked.   However, other Windows-based applications are permitted to continue executing while the **rdoResultset** is being created.

RemoteData Control Asynchronous Operation

If you set the **Options** property to **rdAsyncEnable** before the **RemoteData** control creates the **rdoResultset**, control returns to your application before the **rdoResultset** contains rows.   Check the **StillExecuting** property of the **RemoteData** control's underlying **rdoResultset** object to determine when the first data row is available.   To cancel the query, use the **rdoResultset**.**Cancel** method.

See Also

AsyncCheckInterval Property
Cancel Method
Execute Method
MoreResults Method
OpenResultset Method
Options Property
rdoResultset Object, **rdoResultsets** Collection
RemoteData Control
StillExecuting Property

Choosing a Cursor Type

See Also

Choosing the right cursor for your application can significantly impact performance and resource management. In many cases, use of the forward-only–type result set is the best choice as it only exposes one row of the result set at a time, and is far easier for RDO to create. However, the forward-only result set is not a cursor, and does not permit access to more than one row at a time. Your choice of cursor depends on how many rows you intend to access, how you need to navigate through the result set, how membership is determined, and how you intend to update the data.

Server-Side Cursor Support

Another important aspect of keyset or dynamic cursors is where the keyset is created. If your server supports server-side cursors, as is the case with Microsoft SQL Server 6.0, you can specify that the cursor keyset is created and maintained on the server. If you use client-side cursors, cursor keysets are downloaded to the workstation and stored in local memory. In many cases, using server-side cursors can significantly improve performance. To enable server-side cursors, set the **rdoDefaultCursorDriver** or **CursorDriver** property.

Selecting a Cursor Type

To select a specific type of **rdoResultset** cursor, set the **RemoteData** control's **ResultsetType** property or the *type* argument of the **OpenResultset** method to:

Resultset type	Constant
Forward-only	(Default) **rdOpenForwardOnly**
Static	**rdOpenStatic**
Keyset	**rdOpenKeyset**
Dynamic	**rdOpenDynamic**

Available Cursor Types

The following table summarizes the four types of **rdoResultset** cursors:

Attribute	Forward-only	Static	Keyset	Dynamic
Updatable	Yes (SS) No (CL)	No (SS) Yes (CL)	Yes	Yes
Membership	Fixed	Fixed	Fixed	Dynamic
Visibility	One row	Cursor	Cursor	Cursor
Move current row	Forward	Anywhere	Anywhere	Anywhere
Result of a join	Yes	Yes	Yes	Yes

Notation: *CL* indicates that support for this cursor is provided by the ODBC cursor library. *SS* indicates support by Microsoft SQL Server.

You can choose the type of **rdoResultset** object you want to create using the *type* argument of the **OpenResultset** method or the **ResultsetType** property of the **RemoteData** control. If you don't specify a *type*, the **RemoteData** control creates a keyset-type **rdoResultset**. When using RDO to create **rdoResultset** objects, the default type is forward-only.

Supported Cursor Types

Not all data sources support every type of cursor. In some cases the ODBC driver cursor library must be used in lieu of server-side cursors − this limits the type of cursors supported. The following table summarizes which type of cursor is supported on several typical data sources and on the **RemoteData** control:

Data source	Forward-only	Static	Keyset	Dynamic
SQL Server 4.2	Yes	Yes/CL	No	No
SQL Server 6.0	Yes	Yes	Yes	Yes

Oracle 7.1	Yes	Yes/CL	No	No
RemoteData control	No	Yes	Yes/DD	No

Notation: *CL* indicates that support for this cursor is provided by the ODBC cursor library. *DD* indicates support is provided subject to support by the ODBC driver.

Cursors and the RemoteData Control

If you create an **rdoResultset** and set the **Resultset** property with this new object, the **ResultsetType** property of the **RemoteData** control is set to the **Type** property of the new **rdoResultset**.

Note When using forward-only, read-only result sets, the **rdoConnection** is held open until the last row of data is accessed. This type of cursor can provide performance improvements over other cursors, but can tie up connection resources.

See Also

CursorDriver Property
OpenResultset Method
rdoDefaultCursorDriver Property
ResultsetType Property
Type Property
Understanding Cursors

Using RemoteData Control Properties

See Also

To create an **rdoResultset** with the **RemoteData** control, you must first describe the type and characteristics of the result set using the **RemoteData** control properties. Once the properties are set at run time, use the **Refresh** method to build a new **rdoResultset** object. If you set the **SQL** property at design time, the **RemoteData** control attempts to build a result set when the form containing the control is first loaded.

The following table lists the properties you can use to describe and create an **rdoResultset**.

Property	Description
Connect	Holds the parameters needed by the ODBC driver manager to establish a connection. It is reset after the connection is established to reflect the parameters used to make the connection.
Connection	Set with the **rdoConnection** created by the **RemoteData** control when the connection is established.
CursorDriver	Determines if local or server-side cursors are to be used. Default value is set by **rdoEngine**.**rdoDefaultCursorDriver**; **rdUseIfNeeded** is the default.
DataSourceName	Determines the ODBC data source name used by the ODBC driver manager to establish the connection.
EditMode	Indicates if **AddNew** or **Edit** have been executed against current row.
ErrorThreshold	Determines severity level of critical errors.
KeysetSize	Determines the number of rows in the cursor keyset. Used only for mixed cursors and should be set to 0 in all other cases.
LockType	Determines the type of concurrency used when **rdoResultset** is updated.
LoginTimeout	Determines how long the ODBC driver manager waits before abandoning a connection attempt.
LogMessages	Sets filename of ODBC log file.
MaxRows	Determines maximum number of rows to be returned by the query.
Options	Sets build option(s). **rdAsyncEnable** enables asynchronous queries.
Password	Sets password used to establish a data source connection.
Prompt	Sets ODBC driver manager behavior when connection arguments are missing.
QueryTimeout	Sets how long the ODBC driver manager waits for the first row of a query to become available.
ReadOnly	Sets connection to permit or prohibit changes to the data.
Resultset	Sets or returns the **rdoResultset** object managed by the **RemoteData** control.
ResultsetType	Sets the type of cursor to create.
RowsetSize	Sets the size of the cursor rowset. Also referred to as the "fat cursor" size.
SQL	Determines the SQL query used to create the **rdoResultset**.
Transactions	Indicates if the **rdoResultset** supports transactions.
UserName	Determines the user name used to log on to the ODBC data source.

See Also

rdoResultset Object, **rdoResultsets** Collection
RemoteData Control
Using Bound Controls with the **RemoteData** Control

action query

An SQL query that changes the underlying data or performs some administrative operation, such as adding new tables or users. An action query returns the number of rows affected rather than a result set.

aggregate function

A function, such as **Count**, **Avg**, or **Sum**, used in a query that calculates values. In writing SQL expressions, you can use SQL aggregate functions to determine various statistics.

alias

In Visual Basic, an alternate name you give to an external procedure to avoid conflict with a Visual Basic keyword, global variable, constant, or a name not allowed by the standard naming conventions.

In SQL, an alternate name you give to a column or expression in a SELECT statement, to make it shorter or more meaningful, or to prevent name conflicts when performing SQL queries using expressions that don't return names, or in a query that references the same table more than once.

base table

A table in a remote database.   You can manipulate the structure of a base table using data definition SQL statements, and you can modify data in a base table using **rdoResultset** objects or action queries.

bookmark

A system-generated value identifying the current row that is contained in an **rdoResultset** object's **Bookmark** property.   If you assign the **Bookmark** property value to a variable and then move to another row, you can make the earlier row current again by assigning the value of the variable to the **Bookmark** property.

Boolean

A **True**/**False** or yes/no value.   Boolean values are usually stored in **Bit** columns in a remote database; however, some data sources don't support this data type directly.

Byte data type

A fundamental data type used to hold small positive integer numbers ranging from 0 to 255.

bound control

A data-aware control that can provide access to a specific column or columns in a data source through a **RemoteData** or **Data** control.   A data-aware control can be bound to a **RemoteData** or **Data** control through its **DataSource** and **DataField** properties.   When a **RemoteData** or **Data** control moves from one row to the next, all bound controls connected to the **RemoteData** or **Data** control change to display data from columns in the current row.   When users change data in a bound control and then move to a different row, the changes are automatically saved in the data source.

Cartesian product

The result of joining two relational tables, producing all possible ordered combinations of rows from the first table with all rows from the second table.

Generally, a Cartesian product results from executing an SQL SELECT statement referencing two or more tables in the FROM clause, and not including a WHERE or JOIN clause that indicates how the tables are to be joined.

case-sensitive

Capable of distinguishing between uppercase and lowercase letters. A case-sensitive search finds text that is an exact match of uppercase and lowercase letters. Such a search would, for instance, treat "ZeroLengthStr" and "zerolengthstr" as different. Case sensitivity is a feature of some database management systems.

column

Defines the data type, size, and other attributes of one column of an **rdoTable** or **rdoResultset**. All columns taken as a set define a row in the database. An individual column contains data related in type and purpose throughout the table; that is, a column's definition doesn't change from row to row.

commit

To accept a pending transaction. If you use transaction processing and begin a transaction, none of the changes made in the transaction will be written to the data source until you commit (accept) the transaction.

copy buffer

A location created by the **rdoResultset** object for the contents of a row that is open for editing. The **Edit** method copies the current row to the copy buffer; the **AddNew** method clears the buffer for a new row; and the **Update** method saves the data from the copy buffer to the data source, replacing the current row or inserting the new row. Any statement that resets or moves the current row pointer, or cancels the edit, will discard the copy buffer.

connect string

A string used to specify a data source and other information, such as user name and password.   The connect string is usually assigned to the **Connect** property of an **rdoConnection** object or **RemoteData** control, or as an argument to the **OpenConnection** method.

criteria

A set of limiting conditions, such as = "*Denmark*" (meaning equal to Denmark) or > *30000*, used in creating a query or filter to show a specific set of rows.

current transaction

All changes made to an **rdoResultset** object or a set of rows in a database after you use the last **BeginTrans** method and before you use the **RollbackTrans** or **CommitTrans** method.

remote data object

An object, such as **rdoConnection**, **rdoTable, rdoResultset**, or **rdoPreparedStatement**, that represents an object used to organize and manipulate data in code.

data page

A portion of the database in which row data is stored. Depending on the size of the rows, a data page may contain more than one row. A data page in most remote databases is 2K bytes.

data type (Remote Data)

The attribute of a variable or column that determines what kind of data it can hold. For example, an **rdoColumn** defined with the **rdTypeCHAR** data type is designed to contain text.

database

A set of data related to a particular topic or purpose. A database contains tables and can also contain queries and table relationships, as well as table and column validation criteria.

database engine

The part of the database management system that retrieves data from and stores data in user and system databases.

rdoConnection object

Represents an open connection to a remote data source.

database management system (DBMS)

Software used to organize, analyze, and modify information stored in a database.   For example, the Microsoft SQL Server is an example of a database management system.

RemoteData control

Provides access to data stored in a remote ODBC data source.   The **RemoteData** control allows you to move from row to row in a result set and to display and manipulate data from the rows using bound controls.

data-definition query

An SQL-specific query that can create, alter, or delete a table, or create or delete an index in a database.

data source

A named Open Database Connectivity (ODBC) resource that specifies the location, driver type, and other parameters needed by an ODBC driver to access a database.

database administration

Activities required to preserve the integrity and security of a database, such as maintaining user permissions and backing up and repairing the database.

default environment

The **rdoEnvironment** object that Visual Basic automatically establishes when your application first references any remote data object.   This **rdoEnvironment** is referenced by **rdoEngine.rdoEnvironments**(0) or simply **rdoEnvironments**(0).

expression (Remote Data)

Any combination of operators, constants, literal values, functions, and names of columns, controls, and properties that evaluates to a single value. You can use expressions as settings for many properties and action arguments, to set criteria or define calculated columns in queries.

column properties

Attributes of a column that describe the data it contains. **Size** and **Type** are examples.

filter

A set of criteria applied to rows in order to create a subset of the rows.

forward scroll

Movement toward the end (EOF) of an **rdoResultset** object.

forward-only-type rdoResultset

An **rdoResultset** object in which rows can be searched only from beginning to end; the current row position can't be moved back toward the first row, and only one row at a time is accessible.   Forward-only–type **rdoResultset** objects are useful for quickly retrieving and processing data.

identifier
An element of an expression that refers to the value of a column or property.

index (Remote Data)

A dynamic cross-reference of one or more table data columns that permits faster retrieval of specific rows from a table.   As rows are added, changed, or deleted, the database management system automatically updates the index to reflect the changes.

initialization file

An ASCII text file used to contain parameters for configuring Windows-based applications or Microsoft Windows itself.   Generally, an initialization file uses the extension .INI and is named after the executable program that uses it.   For example, a program named TESTING.EXE would expect an initialization file called TESTING.INI.

Integer data type

A fundamental data type that holds integer numbers. An **Integer** variable is stored as a 16-bit (2-byte) number ranging in value from -32,768 to 32,767. The type-declaration character is **%** (ANSI character 37).

locked

The condition of a data page or row that makes it read-only to all users except the one who is currently entering data in it.

ODBC (Open Database Connectivity)

A standard protocol that permits applications to connect to a variety of external database servers or files. ODBC drivers used by the ODBC driver manager permit access to SQL Server and several other data sources, including text files and Microsoft Excel spreadsheets.

optimistic

A type of locking in which the data page containing one or more rows, including the row being edited, is unavailable to other users only while the row is being updated by the **Update** method, but is available between the **Edit** and **Update** methods. Optimistic locking is used when the **rdConcurRowver** or **rdConcurValues LockType** is used when opening an **rdoResultset**.

parameter

An element containing a value that you can change to affect the results of the query. For example, a query returning data about an employee might have a parameter for the employee's name. You can then use one **rdoParameter** of the **rdoPreparedStatement** object to find data about any employee by setting the parameter to a specific name before running the query.

pessimistic

A type of locking in which the page containing the row being edited is unavailable to other users when you use the **Edit** method and remains unavailable until you use the **Update** method.   Pessimistic locking is enabled when the **rdConcurLock LockType** is used when opening an **rdoResultset**.

query

A formalized instruction to a database to either return a set of rows or perform a specified action on a set of rows, as specified in the query.   For example, the following SQL query statement returns rows:

```
SELECT CompanyName FROM Publishers WHERE State = 'NY'
```

You can create and run select, action, parameter, and stored procedure queries.

read-only

A type of access to data whereby information can be retrieved but not modified.   This will provide better performance in most cases.

row

A set of related data about a person, place, event, or some other item. Table data is stored in rows in the database. Each row is composed of a set of related columns −each column defining one attribute of information for the row. Taken together, a row defines one specific unit of retrievable information in a database.

requery

To rerun a query to reflect changes to the rows, retrieve newly added rows, and eliminate deleted rows.

security

Used to specify or restrict the access that specified users or user groups have to data and objects in a database.

server

The database management system designed to share data with client applications; servers and clients are often connected over a network. A database server usually contains and manages a central repository of data that remote client applications can retrieve and manipulate.

SQL statement

An expression that defines a Structured Query Language (SQL) command, such as SELECT, UPDATE, or DELETE, and which might include clauses such as WHERE and ORDER BY.   SQL strings and statements are typically used in queries and **rdoResultset** objects but can also be used to create or modify a database structure.

The syntax for SQL statements is dependent on the data source.

table

A basic unit of data storage in a relational database. A table stores data in rows and columns and is usually about a particular category of things, such as employees or parts. Also called a base table.

rdoTable object

Represents the stored definition of a base table or an SQL view.

transaction

A series of changes made to a database's data. Mark the beginning of a transaction with the **BeginTrans** statement, commit the transaction using the **CommitTrans** statement, or undo all your changes since **BeginTrans** using the **RollbackTrans** statement.

Transactions are optional, but can increase the speed of operations and allow changes to be reversed.

Transactions can be managed at the **rdoConnection** level or at the **rdoEnvironment** level.

update

The process that saves changes to data in a row. Until the row is saved, changes are stored in a temporary row called the copy buffer.

The UPDATE clause in an SQL statement changes data values in one or more rows in a database table.

message

A packet of information passed from one application to another.

multiuser database

A database that permits more than one user to access and modify the same set of data at the same time. In some cases, the additional "user" may be another instance of your application, or another application running on your system that accesses the same data as some other application.

normalize

To minimize the duplication of information in a relational database through effective table design.

null

A value that indicates missing or unknown data. **Null** values can be entered in columns for which information is unknown and in expressions and queries. In Visual Basic, the **Null** keyword indicates a **Null** value.

null column

A column containing no characters or values. A null column isn't the same as a zero-length string ("") or a column with a value of 0. A column is set to null when the content of the column is unknown. For example, a Date_Completed column in a task table would be left null until a task is completed.

ODBC driver

A dynamic-link library (DLL) used to connect a specific Open Database Connectivity data source with a client application.

parameter query

A query that requires you to provide one or more criteria values, such as Redmond for City, before the query is run.   A parameter query isn't, strictly speaking, a separate kind of query; rather, it extends the flexibility of other queries.

parse

To identify the parts of a statement or expression and then validate those parts against the appropriate language rules.

permission

One or more attributes that specify what kind of access a user has to data or objects in a database.   For example, a table or query with Read Only permission permits a user to retrieve but not edit data in the table or query.

rowset population

The process of loading **rdoResultset** rows into memory.

rdoResultset objects populate the number of rows defined by the **RowsetSize** attribute. If you are using server-side cursors, only this number of rows is present in memory at any given time.

session

A session begins when a user connects to a data source and ends when a user disconnects. All operations performed during a session are subject to permissions determined by the login user name and password.   Sessions are implemented as **rdoConnection** objects.

Single data type

A fundamental data type that holds single-precision floating-point numbers in IEEE format. A **Single** variable is stored as a 32-bit (4-byte) number ranging in value from -3.402823E38 to -1.401298E-45 for negative values, from 1.401298E-45 to 3.402823E38 for positive values, and 0.   The type-declaration character is **!**.

Structured Query Language (SQL)

A language used in querying, updating, and managing relational databases. SQL can be used to retrieve, sort, and filter specific data to be extracted from the database.

SQL database

A database that can be accessed through the use of Open Database Connectivity (ODBC) data sources or another interface native to the database. Also known as a relational database.

SQL-specific query
A query that can be created only by writing an SQL statement.

String data type

A fundamental data type that holds character information. A **String** variable can contain approximately 65,535 bytes (64K), is either fixed-length or variable-length, and contains one character per byte. Fixed-length strings are declared to be a specific length. Variable-length strings can be any length up to 64K, less a small amount of storage overhead.

The type-declaration character for the **String** data type is **$**.

temporary disk

The directory identified by the TEMP operating system environment variable.   Also known as temporary drive.   Although the TEMP environment variable may point to a RAM disk, this isn't recommended.

TEMP

A TEMP environment variable is initialized by your system when it is started. Generally, TEMP points to an area on your hard disk used by Microsoft Windows and other programs to store information that doesn't need to be saved after you shut down your system. For example, the following line in your AUTOEXEC.BAT file points the TEMP environment variable to the D:\TEMPAREA directory:

```
SET TEMP=D:\TEMPAREA
```

update query

An action query that changes base table data according to criteria you specify.   An update query doesn't return any rows, but it does return the number of rows affected.

user account

An account identified by a user name and password that is created to manage access to objects in a remote database.

validation

The process of checking whether entered data meets certain conditions or limitations.

WHERE clause

The part of an SQL statement that specifies which rows to retrieve.   The WHERE clause limits the scope of the query and specifies which columns are used to join multiple tables.

Yes/No data type

A column data type that contains a **Boolean** (**True**/**False** or yes/no) value.

zero-length string

A string containing no characters. The **Len** function of a zero-length string returns 0.

DDL (Data Definition Language)

The language used to describe attributes of a database, especially the schema associated with tables, columns, and storage strategy.

ODBC data source

A database or database server used as a source of data. ODBC data sources are referred to by their Data Source Name. Data sources can be registered using either the ODBC Administrator in the Windows Control Panel or the **rdoRegisterDataSource** method.

reserved word

A word that is part of the data source SQL language. Reserved words include the names of statements, predefined functions and data types, methods, operators, and objects. Examples include SELECT, UPDATE, BETWEEN, SET, and INSERT.

string expression

Any expression that evaluates to a sequence of contiguous characters.   Elements of the expression can include a function that returns a string, a string literal, a string constant, a string variable, a string **Variant**, or a function that returns a string **Variant** (**VarType** 8).

Long data type

A four-byte integer (a whole number between -2,147,483,648 and 2,147,483,647, inclusive).

numeric expression

Any expression that can be evaluated as a number. Elements of the expression can include any combination of keywords, variables, constants, and operators that result in a number.

Legend

▬ Applies only to object.

☐ Applies only to collection.

▣ Applies to both object and collection.

Object Browser

A dialog box that lets you examine the contents of an object library to get information about the objects provided, their methods and properties, and possibly their constants.

object library

A file with the .OLB extension that provides information to OLE Automation clients (like Visual Basic) about available OLE Automation objects. You can use the Object Browser to examine the contents of an object library to get information about the objects provided.

ASCII character set

American Standard Code for Information Interchange (ASCII) 7-bit character set widely used to represent letters and symbols found on a standard U.S. keyboard.   The ASCII character set is the same as the first 128 characters (0▪127) in the ANSI character set.

column data types

The following table describes the column data types.

Column data type	Description
rdTypeCHAR	Fixed-length character string. Length set by **Size** property.
rdTypeNUMERIC	Signed, exact, numeric value with precision p and scale s (1 p 15; 0 s p).
rdTypeDECIMAL	Signed, exact, numeric value with precision p and scale s (1 p 15; 0 s p).
rdTypeINTEGER	Signed, exact numeric value with precision 10, scale 0 (signed: -231 n 231-1; unsigned: 0 n 232-1).
rdTypeSMALLINT	Signed, exact numeric value with precision 5, scale 0 (signed: -32,768 n 32,767, unsigned: 0 n 65,535).
rdTypeFLOAT	Signed, approximate numeric value with mantissa precision 15 (zero or absolute value 10-308 to 10308).
rdTypeREAL	Signed, approximate numeric value with mantissa precision 7 (zero or absolute value 10-38 to 1038).
rdTypeDOUBLE	Signed, approximate numeric value with mantissa precision 15 (zero or absolute value 10-308 to 10308).
rdTypeDATE	Date ■ data source dependent.
rdTypeTIME	Time ■ data source dependent.
rdTypeTIMESTAMP	TimeStamp ■ data source dependent.
rdTypeVARCHAR	Variable-length character string. Maximum length 255.
rdTypeLONGVARCHAR	Variable-length character string. Maximum length determined by data source.
rdTypeBINARY	Fixed-length binary data. Maximum length 255.
rdTypeVARBINARY	Variable-length binary data. Maximum length 255.
rdTypeLONGVARBINARY	Variable-length binary data. Maximum data source dependent.
rdTypeBIGINT	Signed, exact numeric value with precision 19 (signed) or 20 (unsigned), scale 0 (signed: -263 n 263-1; unsigned: 0 n 264-1).
rdTypeTINYINT	Signed, exact numeric value with precision 3, scale 0 (signed: -128 n 127, unsigned: 0 n 255).
rdTypeBIT	Single binary digit.

current row

The row in an **rdoResultset** object that you can use to modify or examine data.   Use the *Move* methods to reposition the current row in a rowset.

Only one row in an **rdoResultset** can be the current row; however, an **rdoResultset** may have no current row.   For example, after the current **rdoResultset** row has been deleted, or when an **rdoResultset** has no rows, the current row is undefined.   In this case, operations that refer to the current row result in a trappable error.

data manipulation language (DML)

The SQL statement properties and methods you use to write applications or queries that access and manipulate the data in existing databases.   This includes facilities for querying the database, navigating through its tables, performing updates, and adding or deleting rows.

Date/Time

Dates and times are stored internally as different parts of a real number.

The value to the left of the decimal represents a date between January 1, 100 and December 31, 9999, inclusive.   Negative values represent dates prior to December 30, 1899.

The value to the right of the decimal represents a time between 0:00:00 and 23:59:59, inclusive.   Midday is represented by .5.

design time

The time during which you build an application in the development environment by adding controls, setting control or form properties, and so on. In contrast, during run time, you interact with the application as a user would.

environment

An **rdoEnvironment** object defines a session for a specific user. When RDO is referenced for the first time, a default **rdoEnvironment** object is created with a password of "" and user name of "".

object expression

An expression that specifies a particular object. This expression can include any of the object's containers.

rdoPreparedStatement object

A remote data object that contains a prepared SQL statement and collection of **rdoParameter** objects for each parameter in the **rdoPreparedStatement**.

result set

The results of a query.   Result sets might contain rows when a query contains a SELECT statement.   Action queries do not return rows but do return result sets that contain information about the operation, such as rows affected.

run time

The time when an application is running. During run time, you interact with the code as a user would. In contrast, design time is when the application is developed.

server-side cursor

Cursor keysets that are created on the server instead of on the client workstation.

sort order

A sequencing principle used to order data, such as dictionary, binary, ascending, descending, and so on.

SQL view

SQL views are similar to queries: both allow you to limit the rows and columns displayed from one or more tables, and both provide similar functionality.   SQL views are logical sets of rows where a table represents the actual rows.

statement handle

Used by the driver to reference storage for names, parameter and binding information, error messages, and other information related to a statement processing stream, such as the **hStmt** property of the **rdoResultset**.

stored procedure

A pre-compiled procedure stored in a data source, available to be called from an application as needed. Predefined queries reduce the overhead of repeatedly specifying the same selection criteria, and are much faster than submitting an ad-hoc query.

two-phase commit

Allows an application to coordinate updates among multiple SQL servers.   This implementation of distributed transactions treats transactions on separate SQL servers as a single transaction. The service uses one SQL server, the commit server, as a record keeper that helps the application determine whether to commit or to roll back transactions. Thus, the two-phase commit guarantees that either all the databases on the participating servers are updated or that none of them are.

Variant data type

A special data type that can contain numeric, string, or date data, as well as the special values **Empty** and **Null**. The **VarType** function defines how the data in a **Variant** is treated. All variables become variant types if not explicitly declared as some other type.

asynchronous

A type of query mode in which SQL queries return immediately, even though the results are still pending.   This enables an application to continue with other processing while the query is pending completion.

cursor

A logical set of rows managed by the data source or ODBC driver manager. The cursor is so named because it indicates the current position in the result set, just as the cursor on a CRT screen indicates current position.

connection handle

Identifies memory storage for information about a particular connection. RDO will request a connection handle prior to connecting to a data source; RDO manages connection handles automatically through the **rdoConnection** object. Each connection handle is associated with an environment handle. An environment handle can have multiple connection handles associated with it, and there can be multiple environments.

Data Access Objects (DAO)

Objects that are defined by the Microsoft Jet database engine.   You use data access objects, such as the **Database**, **TableDef**, **Recordset**, and **QueryDef** objects, to represent objects that are used to organize and manipulate data in code.

dynamic-link library (DLL)
A library of routines loaded and linked to applications at run time.

dynamic-type rdoResultset

The result of a query that can have updatable rows. A dynamic-type **rdoResultset** is a dynamic set of rows that you can use to add, change, or delete rows from an underlying database table or tables. A dynamic-type **rdoResultset** can contain columns from one or more tables in a database. Membership of a dynamic **rdoResultset** is not fixed.

environment handle

Identifies memory storage for global information, including the valid connection handles and current active connection handle. RDO will request this handle prior to connecting to a data source. The remote data objects manage environment handles automatically through the **rdoEnvironment** object.

escape codes

Allow you to specify a value such as a date or time, in a data-independent way. For example, {d 'value'} allows you to specify the date (SELECT * FROM table WHERE DateField = {d "2/17/94"}). When this query is submitted to the ODBC driver, it will scan the string and replace the escape clause with the date in the proper form for the specific ODBC driver you are using.

keyset

The set of key values that are stored on the server or ODBC's cursor library.

keyset-type rdoResultset

The result of a query that can have updatable rows.   Movement within the keyset is unrestricted. A keyset-type **rdoResultset** is a set of rows that you can use to add, change, or delete rows from an underlying database table or tables.   A keyset-type **rdoResultset** can contain columns from one or more tables in a database. Membership in a keyset **rdoResultset** is fixed.

native error

An error generated and returned from the database management system of the data source on a given connection.

ODBC driver manager
Provides the interface from the host language to the specific back-end data source driver.

Remote Data Objects (RDO)

Provide an information model for accessing remote data sources through ODBC. RDO offers a set of objects that make it easy to connect to a database, execute queries and stored procedures, manipulate results, and commit changes to the server.

scope

Defines the visibility of a variable, procedure, or object.   For example, a variable declared as **Public** is visible to all procedures in all modules.   Variables declared in procedures are visible only within the procedure and lose their value between calls unless they are declared **Static**.

static-type rdoResultset

The membership, order, and values in a result set used by a static cursor are generally fixed when the cursor is opened. Rows updated, deleted, or inserted by other users are not detected by the cursor until it is closed and then reopened.

timestamp
Contains a unique value that is updated whenever a row is updated.

procedural query

An SQL query that executes a stored procedure.

connection

A link to an ODBC data source.

compile time

The moment at which source code is translated into executable code.

select query

A query that asks a question about the data stored in your tables, and returns an **rdoResultset** object without changing the data. Once the **rdoResultset** data is retrieved, you can examine and make changes to the data in the underlying queries.

Error Event (Remote Data)

See Also Applies To

Occurs only as the result of a data access error that takes place when no Visual Basic code is being executed.

Syntax

Private Sub *object* **_Error(**[*index* **As Integer**,]*Number* **As Long,** *Description* **As String,** *Scode* **As Long,** *Source* **As String,** *HelpFile* **As String,** *HelpContext* **As Long,** *CancelDisplay* **As Boolean)**

The Error event syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.
index	Identifies the control if it's in a control array.
Number	The native error number.
Description	Describes the error.
Scode	ODBC error return code.
Source	Source of the error.
HelpFile	The path to a Help file containing more information on the error.
HelpContext	The Help file context number.
CancelDisplay	A number corresponding to the action you want to take, as described in Settings.

Settings

The settings for *CancelDisplay* are:

Constant	**Value**	**Description**
rdDataErrContinue	0	Continue.
rdDataErrDisplay	1	(Default) Display the error message.

Remarks

Generally, the Error event arguments correspond to the properties of the **rdoError** object.

You usually provide error-handling functionality for run-time errors in your code. However, run-time errors can occur when none of your code is running, as when:

- A user clicks a **RemoteData** control button.
- The **RemoteData** control automatically opens an **rdoConnection** and creates an **rdoResultset** object after the Form_Load event.
- A custom control performs an operation, such as the **MoveNext** method, the **AddNew** method, or the **Delete** method.

If an error results from one of these actions, the Error event occurs.

If you don't code an event procedure for the Error event, Visual Basic displays the message associated with the error.

Close

Error Event (Remote Data) Applies To

RemoteData Control

See Also

AddNew Method
Delete Method
MoveFirst, **MoveLast**, **MoveNext**, **MovePrevious** Methods
rdoError Object, **rdoErrors** Collection
rdoResultset Object, **rdoResultsets** Collection
RemoteData Control

Reposition Event (Remote Data)

See Also Applies To

Occurs after a row becomes the current row.

Syntax

Private Sub *object***.Reposition (**[*index* **As Integer**]**)**

The Reposition event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	Identifies the control if it's in a control array.

Remarks

When a **RemoteData** control is loaded, the first row in the **rdoResultset** object becomes the current row, causing the Reposition event. Whenever a user clicks any button on the **RemoteData** control (moving from row to row or using one of the *Move* methods, such as **MoveNext**, or any other property or method that changes the current row), the Reposition event occurs after each row becomes current.

In contrast, the Validate event occurs before moving to a different row.

You can use this event to perform calculations based on data in the current row or to change the form in response to data in the current row.

Close

Reposition Event (Remote Data) Applies To

RemoteData Control

See Also

AbsolutePosition Property
Error Event
Move Method
MoveFirst, **MoveLast**, **MoveNext**, **MovePrevious** Methods
PercentPosition Property
rdoResultset Object, **rdoResultsets** Collection
Validate Event

Validate Event (Remote Data)

See Also Applies To

Occurs before a different row becomes the current row; before the **Update** method (except when data is saved with the **UpdateRow** method); and before a **Delete**, **Unload**, or **Close** operation.

Syntax

Private Sub *object*_**Validate (**[*index* **As Integer**,] *action* **As Integer**, *save* **As Integer)**

The Validate event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	Identifies the control if it's in a control array.
action	An **Integer** or constant that indicates the operation causing this event to occur, as described in Settings.
save	A **Boolean** expression that specifies whether bound data has changed, as described in Settings.

Settings

The settings for *action* are:

Constant	Value	Description
rdDataActionCancel	0	Cancel the operation when the **Sub** exits.
rdDataActionMoveFirst	1	**MoveFirst** method.
rdDataActionMovePrevious	2	**MovePrevious** method.
rdDataActionMoveNext	3	**MoveNext** method.
rdDataActionMoveLast	4	**MoveLast** method.
rdDataActionAddNew	5	**AddNew** method.
rdDataActionUpdate	6	**Update** operation (not **UpdateRow**).
rdDataActionDelete	7	**Delete** method.
rdDataActionBookmark	8	The **Bookmark** property has been set.
rdDataActionClose	9	The **Close** method.
rdDataActionUnload	10	The form is being unloaded.

The settings for *save* are:

Setting	Description
True	Bound data has changed.
False	Bound data has not changed.

Remarks

The *save* argument initially indicates whether bound data has changed.   This argument can still be **False** if data in the copy buffer is changed.   If *save* is **True** when this event exits, the **Edit** and **UpdateRow** methods are invoked.

This event occurs if no changes have been made to data in bound controls, and even if no bound controls exist.   You can use this event to change values and update data.   You can also choose to save data or stop whatever action is causing the event to occur and substitute a different action.

You can change the *action* argument to convert one action into another.   You can change the various *Move* methods and the **AddNew** method, which can be freely exchanged (any *Move* into **AddNew**, any *Move* into any other *Move*, or **AddNew** into any *Move*). Attempting to change **AddNew** or one of the *Moves* into any of the other actions is either ignored or produces a trappable error.   Any action can be stopped by setting *action* to **rdDataActionCancel**.

In your code for this event, you can check the data in each bound control where

DataChanged is **True**.   You can then set **DataChanged** to **False** to avoid saving that data in the database.

You can't use any methods (such as **MoveNext**) on the underlying **rdoResultset** object during this event.

Close

Validate Event (Remote Data) Applies To

RemoteData Control

See Also

AddNew Method
Bookmark Property
Close Method
Delete Method
Edit Method
EditMode Property
MoveFirst, **MoveLast**, **MoveNext**, **MovePrevious** Methods
rdoResultset Object, **rdoResultsets** Collection
RemoteData Control
Update Method
UpdateRow Method

QueryCompleted Event (Remote Data)

See Also Applies To

Occurs after the query of an **rdoResultset** returns the first result set.

Syntax

Private Sub *object***.QueryCompleted (**[*index* **As Integer**]**)**

The QueryCompleted event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	Identifies the control if it's in a control array.

Remarks

When a **RemoteData** control completes the asynchronous creation of an **rdoResultset**, the QueryCompleted event is invoked. This event is not triggered if you execute the **Cancel** method which terminates processing of the query.

See Also

AsyncCheckInterval Property
Cancel Method
rdoResultset Object, **rdoResultsets** Collection
StillExecuting Property

Close

QueryCompleted Event (Remote Data) Applies To

RemoteData Control

Remote Data Objects and the RemoteData Control

Remote Data Objects Language Summary

An alphabetic listing of all programming language topics

Constants	Objects and Collections
Events	Properties
Methods	Trappable Errors
Conceptual Topics	Glossary

Controls

RemoteData Control

Object Models

Remote Data Object Model

Glossary (Remote Data)

Glossary (Remote Data)

A
▪
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
See Also

A

action query
aggregate function
alias
ASCII character set
asynchronous

B

base table
bookmark
Boolean
bound control
Byte data type

C

Cartesian product
case-sensitive
column
column data types
column properties
commit
compile time
connect string
connection
connection handle
copy buffer
criteria
current row
current transaction
cursor

D

Data Access Objects (DAO)
data manipulation language (DML)
data page
data source
data type (Remote Data)
data-definition query
database
database administration
database engine
database management system (DBMS)
Date/Time
DDL (Data Definition Language)
default environment
design time
dynamic-link library (DLL)
dynamic-type **rdoResultset**

E

environment
environment handle
escape codes
expression (Remote Data)

F-J

filter
forward scroll
forward-only-type **rdoResultset**
identifier
index (Remote Data)
initialization file

Integer data type

K

keyset
keyset-type **rdoResultset**
locked

L

Long data type

M

message
multiuser database

N

native error
normalize
null
null column
numeric expression

O

Object Browser
object expression
object library
ODBC (Open Database Connectivity)
ODBC data source
ODBC driver
ODBC driver manager
optimistic

P

parameter
parameter query
parse
permission
pessimistic
procedural query

Q

query

R

rdoConnection object
rdoPreparedStatement object
rdoTable object
read-only
remote data object
Remote Data Objects (RDO)
RemoteData control

requery
reserved word
result set
row
rowset population
run time

S

scope
security
select query
server
server-side cursor
session
Single data type
sort order
SQL database
SQL statement
SQL view
SQL-specific query
statement handle
static-type **rdoResultset**
stored procedure
String data type
string expression
Structured Query Language (SQL)

T

table
TEMP
temporary disk
timestamp
transaction
two-phase commit

U

update
update query
user account

V

validation
Variant data type

W-Z

WHERE clause
Yes/No data type
zero-length string

See Also

Remote Data Objects and the RemoteData Control

Close

Conceptual Information Summary (Remote Data)

Close

Conceptual Information Summary (Remote Data)

See Also

Choosing a Cursor Type
Creating Parameter Queries
Remote Data Object Model
Remote Data Objects and Collections
Remote Data Objects Overview
Repositioning the Current Row Pointer
Running Asynchronous Queries
Understanding Cursors
Understanding **rdoResultset** Objects
Using Bound Controls with the **RemoteData** Control
Using ODBC Handles
Using **RemoteData** Control Properties
Using the **RemoteData** Control

See Also

Remote Data Objects and the RemoteData Control

Close

Remote Data Objects Language Summary

See Also

Remote Data Objects and the RemoteData Control

Close

Remote Data Objects Method Summary

*

See Also

Grouped by Object or Collection

rdoColumn, rdoColumns
rdoConnection, rdoConnections
rdoEngine
rdoEnvironment, rdoEnvironments
rdoError, rdoErrors
rdoParameter, rdoParameters
rdoPreparedStatement, rdoPreparedStatements
rdoResultset, rdoResultsets
rdoTable, rdoTables

Listed Alphabetically

A

AddNew
AppendChunk

B

BeginTrans

C

Cancel
CancelUpdate
Clear
Close
ColumnSize

See Also

Remote Data Objects and the RemoteData Control

Remote Data Objects Object and Collection Summary

Close

Remote Data Objects Object and Collection Summary

See Also

rdoColumn, rdoColumns
rdoConnection, rdoConnections
rdoEngine
rdoEnvironment, rdoEnvironments
rdoError, rdoErrors
rdoParameter, rdoParameters
rdoPreparedStatement, rdoPreparedStatements
rdoResultset, rdoResultsets
rdoTable, rdoTables

See Also

Remote Data Objects and the RemoteData Control

Remote Data Objects Property Summary

Close

Remote Data Objects Property Summary

*

See Also

This reference groups all remote data properties by object or collection and lists them alphabetically.

Grouped by Object or Collection

rdoColumn, rdoColumns
rdoConnection, rdoConnections
rdoEngine
rdoEnvironment, rdoEnvironments
rdoError, rdoErrors
rdoParameter, rdoParameters
rdoPreparedStatement, rdoPreparedStatements
rdoResultset, rdoResultsets
rdoTable, rdoTables

Listed Alphabetically

A

AbsolutePosition
AllowZeroLength
AsyncCheckInterval
Attributes

B

BindThreshold
BOF
BOFAction
Bookmark

P

Password
PercentPosition
Prompt

Q

QueryTimeout

R

rdoDefaultCursorDriver
rdoDefaultErrorThreshold
rdoDefaultLoginTimeout
rdoDefaultPassword
rdoDefaultUser
rdoLocaleID
rdoVersion
ReadOnly
Required
Restartable
Resultset
ResultsetType
RowCount
RowsAffected
RowsetSize

S

Size
Source
SourceColumn
SourceTable
SQL
SQLRetCode
SQLState
StillExecuting

T

Transactions
Type

U

Updatable
UserName

V-Z

Value
Version

See Also

Remote Data Objects and the RemoteData Control

Remote Data Objects Event Summary

Close

Remote Data Objects Event Summary

See Also

This reference lists all **RemoteData** control events alphabetically.

Error
QueryCompleted
Reposition
Validate

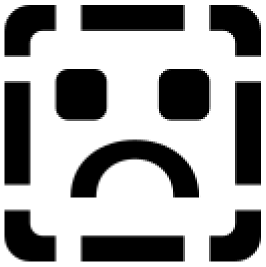
See Also

Remote Data Objects and the RemoteData Control
Remote Data Method Summary
Remote Data Property Summary

Remote Data Object Model

See Also

The following diagram illustrates the Remote Data Object model.   Click a specific object in the diagram to get more information.



See Also

RemoteData Control
Remote Data Objects Overview
Remote Data Objects and Collections
Using the **RemoteData** Control

Close

rdoColumn Methods

Legend

AppendChunk–

ColumnSize–

GetChunk–

Item–

Close

rdoConnection Methods

Legend

BeginTrans–

Cancel–
Close–
CommitTrans–
CreatePreparedStatement–
Execute–
Item–
OpenResultset–
RollbackTrans–

Close

rdoEnvironment Methods

Legend

BeginTrans–
Close–
CommitTrans–
Item–
OpenConnection–
RollbackTrans–

Close

rdoPreparedStatement Methods

Legend

Cancel–

Close–

Execute–

Item–

OpenResultset–

–

rdoResultset Methods

Legend

AddNew–

Cancel–
CancelUpdate–
Close–
Delete–
Edit–
GetRows–
Item–
MoreResults–
Move–
MoveFirst–
MoveLast–
MoveNext–
MovePrevious–

Requery–
Update–

rdoTable Methods

Legend

Item

OpenResultset

Refresh

–

rdoColumn Properties

Legend

AllowZeroLength–

Attributes–
ChunkRequired–
Count–
Name–
OrdinalPosition–
Required–
Size–
SourceColumn–
SourceTable–
Type–
Updatable–
Value–

–

rdoConnection Properties

Legend

AsyncCheckInterval–
Connect–
Count–
hDbc–
Name–
QueryTimeout–
RowsAffected–
StillExecuting–
Transactions–
Updatable–
Version–

–

rdoEngine Properties

Legend

rdoDefaultCursorDriver–
rdoDefaultErrorThreshold–
rdoDefaultLoginTimeout–
rdoDefaultPassword–
rdoDefaultUser–
rdoVersion–

rdoEnvironment Properties

Legend

Count–

CursorDriver–
hEnv–
LoginTimeout–
Name–
Password–
UserName–

–

rdoError Properties

Legend

Count–

Description–
HelpContext–
HelpFile–
Number–
Source–
SQLRetCode–
SQLState–

–

rdoParameter Properties

Legend

Count–

Direction–
Name–
Type–
Value–

rdoParameter Methods

Legend

Item

–
# rdoPreparedStatement Properties

Legend

BindThreshold–
Connect–
Count–
ErrorThreshold–
hStmt–
KeysetSize–
LockType–
LogMessages–
MaxRows–
Name–
QueryTimeout–
RowsAffected–
RowsetSize–
SQL–
StillExecuting–
Type–
Updatable–

–

rdoResultset Properties

Legend

AbsolutePosition–

BOF–
Bookmark–
Bookmarkable–
Count–
EOF–
hStmt–
LastModified–
LockEdits–
Name–
PercentPosition–
Restartable–
RowCount–
StillExecuting–
Transactions–

Type–
Updatable–

–

rdoTable Properties

Legend

Count–

Name–
RowCount–
Type–
Updatable–

rdoEngine Methods

Legend

rdoCreateEnvironment

rdoRegisterDataSource

Remote Data Trappable Errors

Remote Data Trappable Errors

See Also

Trappable errors can occur while an application is running, either within the Visual Basic environment or as a stand-alone executable.   Some of these can also occur during design time or compile time.   You can test and respond to trappable errors using the **On Error** statement and the **rdoError** object's **Number** property.

Remote Data Control (RDC) Messages

Remote Data Object (RDO) Messages

See Also

Remote Data Objects and the RemoteData Control

RemoteData Control (RDC) Messages

See Also

Code	Message
40500	An unexpected internal error has occurred
40501	An unexpected error occurred
40502	An error has occurred. Unable to retrieve error information
40503	A control canceled the operation or an unexpected internal error has occurred
40504	Could not refresh controls
40505	Invalid property value
40506	Invalid object
40507	Method cannot be called in RDC's current state
40508	One or more of the arguments is invalid
40509	Resultset is empty
40510	Out of memory
40511	Resultset not available
40512	The connection is not open
40513	Property cannot be set in RDC's current state
40514	Property not available in RDC's current state
40515	Type mismatch
40516	Cannot connect to Remote Data Object

See Also

Remote Data Object (RDO) Messages

Remote Data Object (RDO) Messages

See Also

Code	Message
40000	An error occurred configuring the DSN. Please check the parameters and try again
40001	SQL returned No Data Found
40002	An internal ODBC error was encountered
40003	An invalid value for the cursor driver was passed
40004	An invalid ODBC handle was encountered
40005	Invalid connection string
40006	An unexpected error occurred
40008	Invalid operation for forward only cursor
40009	No current row
40010	Invalid row for AddNew
40011	Object is invalid or not set
40014	Incompatible data types for compare
40016	An error occurred loading the version library (VERSION.DLL)
40017	Can't execute unprepared rdoPreparedStatement
40018	Can't execute empty rdoPreparedStatement
40019	An invalid value for the concurrency option was passed
40021	Object Collection: This collection doesn't support location by text tag
40022	The rdoResultset is empty
40023	Invalid state for Move
40024	Already beyond the end of the resultset
40025	BOF already set
40026	Invalid resultset state for update
40027	Invalid bookmark
40028	Invalid bookmark argument to move
40029	Can't move relative to current row as EOF/BOF already set
40032	An error occurred loading the ODBC installation library (ODBCCP32.DLL)
40033	An invalid value for the prompt option was passed
40034	An invalid value for the cursor type parameter was passed
40035	Column not bound correctly
40036	Unbound column - use GetChunk
40037	Can't assign value to unbound column
40038	Can't assign value to non-updatable column
40039	Can't assign value to column unless in edit mode
40040	Incorrect type for parameter
40041	Object Collection: Couldn't find item indicated by text
40042	Can't assign value to unbound parameter
40043	Can't assign value to output-only parameter
40045	You cannot execute a query when an asynchronous query is in progress
40046	The object has already been closed
40047	You must specify a valid name for the environment
40048	This environment name already exists in the collection
40049	Object collection: illegal modification -- collection is read-only
40050	GetNewEnum: Couldn't get interface for IID_IUnknown

40054	An invalid parameter was passed
40055	Invalid operation
40056	The row you attempted to move to has been deleted
40057	An attempt was made to issue a select statement using the Execute method
40058	The resultset is read only
40059	The user canceled the operation

See Also

RemoteData Control (RDC) Messages

rdoError Methods

Legend

Clear

Item

AddNew Method (Remote Data)

See Also Applies To

Creates a new row for an updatable **rdoResultset** object.

Syntax

*object***.AddNew**

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

Remarks

The **AddNew** method prepares a new row you can edit and subsequently add to the **rdoResultset** object named by *object* using the **Update** method. This method initializes the columns to **Null**.

After you modify the new row, use the **Update** method to save the changes and add the row to the result set. No changes are made to the database until you use the **Update** method. The **AddNew** method does not return an error if the **rdoResultset** is not updatable. A trappable error is triggered when the **Update** method is used against an object that is not updatable. For an object to be updatable, the **rdoColumn**, **rdoResultset**, and **rdoConnection** objects must all be updatable–check the **Updatable** property of each of these objects before performing an update.

Caution If you use the **AddNew** method on a row and then perform any operation that moves to another row without using **Update**, your changes are lost without warning. In addition, if you close the *object* or end the procedure which declares the *object* or its **rdoConnection** object, the new row and the changes made to it are discarded without warning.

A newly added row might be visible as a part of the **rdoResultset** if your data source and type of cursor support it. For example, newly added rows are not included in a static-type **rdoResultset**. In some cases, if you add a row to an **rdoResultset** using the **AddNew** method on the result set, the row is visible in the **rdoResultset** and included in the underlying table where it becomes visible to any new **rdoResultset** objects.

When newly added rows are included in the **rdoResultset**, the row that was current before you used **AddNew** remains current. If you want to make the new row current, you can set the **Bookmark** property to the bookmark identified by the **LastModified** property setting.

If you need to cancel a pending **AddNew** operation, use the **CancelUpdate** method.

See Also

BeginTrans, **CommitTrans**, **RollbackTrans** Methods
Bookmark Property
CancelUpdate Method
Delete Method
Edit Method
LastModified Property
Move Method
MoveFirst, **MoveLast**, **MoveNext**, **MovePrevious** Methods
rdoConnection Object, **rdoConnections** Collection
Updatable Property
Update Method

AddNew Method (Remote Data) Applies To

rdoResultset Object

AppendChunk Method (Remote Data)

See Also Applies To

Appends data from a **Variant** expression to an **rdoColumn** object with a data type of **rdTypeLONGVARBINARY** or **rdTypeLONGVARCHAR**.

Syntax

object!*column*.**AppendChunk** *source*

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

Part	Description
object	An object expression that evaluates to the **rdoResultset** object containing the **rdoColumns** collection.
column	An object expression that evaluates to an **rdoColumn** object whose **ChunkRequired** property is set to **True**.
source	A string expression or variable containing the data you want to append to the **rdoColumn** object specified by *column.*

Remarks

Chunk data columns are designed to store binary or text values that can range in size from a few characters to over 1.2GB and are stored in the database on successive data pages. In most cases, *chunk* data cannot be managed with a single operation, so you must use the *chunk* methods to save and write data. If the **ChunkRequired** property is **True** for a column, you must use the **AppendChunk** method to manipulate column data.

Use the **AppendChunk** method to write successive blocks of data to the database column and **GetChunk** to extract data from the database column. Certain operations (copying, for example) involve temporary strings. If string space is limited, you may need to work with smaller segments of a *chunk* column instead of the entire column.

Use the **BindThreshold** property to specify the largest column size that will be automatically bound.

Use the **ColumnSize** property to determine the number of bytes in a *chunk* column. Note that for variable-sized columns, it is not necessary to write back the same number of bytes as returned by the **ColumnSize** property as **ColumnSize** reflects the size of the column before changes are made.

If there is no current row when you use **AppendChunk**, a trappable error occurs.

Note The initial **AppendChunk** (after the first **Edit** method), even if the row already contains data, replaces existing column data. Subsequent **AppendChunk** calls within a single **Edit** session appends data to existing column data.

See Also

BindThreshold Property
ChunkRequired Property
ColumnSize Method
GetChunk Method
rdoColumn Object, **rdoColumns** Collection
rdoResultset Object, **rdoResultsets** Collection
Type Property

_

AppendChunk Method (Remote Data) Applies To

rdoColumn Object

BeginTrans, CommitTrans, RollbackTrans Methods (Remote Data)

See Also 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 **rdoPreparedStatement** 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 unit.

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 aren't 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.

If you use the **CommitTrans** or **RollbackTrans** method without first using the **BeginTrans** method, an error occurs.

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.

See Also

Close Method
rdoCreateEnvironment Method
rdoConnection Object, **rdoConnections** Collection
rdoDefaultUser, rdoDefaultPassword Properties
rdoEnvironment Object, **rdoEnvironments** Collection
rdoResultset Object, **rdoResultsets** Collection
Refresh Method
Transactions Property

BeginTrans, CommitTrans, RollbackTrans Methods (Remote Data) Apply To

rdoConnection Object
rdoEnvironment Object
RemoteData Control

Cancel Method (Remote Data)

See Also Applies To

Cancels the processing of a query running in asynchronous mode, or cancels any pending results against the specified **rdoResultset** 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 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.

You can also use the **Cancel** method against a synchronous **rdoResultset** 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.

See Also

CancelUpdate Method
Execute Method
MoreResults Method
OpenResultset Method

_

Cancel Method (Remote Data) Applies To

rdoConnection Object
rdoPreparedStatement Object
rdoResultset Object
RemoteData Control

CancelUpdate Method (Remote Data)

See Also 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 hasn't 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 doesn't change, and various properties, such as **BOF** and **EOF**, aren't updated.

_

CancelUpdate Method (Remote Data) Applies To

rdoResultset Object

See Also

AddNew Method
BOF, **EOF** Properties
Cancel Method
Edit Method
EditMode Property
rdoResultset Object, **rdoResultsets** Collection
Update Method

Close Method (Remote Data)

See Also 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. For example, using the **Close** method on an **rdoResultset** removes it from the **rdoResultsets** collection. 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 **rdoPreparedStatement** 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.

See Also

Delete Method
OpenResultset Method
Update Method

Close Method (Remote Data) Applies To

rdoConnection Object
rdoEnvironment Object
rdoPreparedStatement Object
rdoResultset Object

ColumnSize Method (Remote Data)

See Also 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
varname	The name of a **Long** or **Variant** variable.
object	An object expression that evaluates to the **rdoResultset** object containing the **rdoColumns** collection.
column	The name of an **rdoColumn** object whose **ChunkRequired** property is set to **True**.

Remarks

When working with data types that span multiple database pages, you must use the *chunk* methods to manage the data.   You must also use the **GetChunk** and **AppendChunk** methods to manage *chunk* data when the **ChunkRequired** property is **True**.

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.

See Also

AppendChunk Method
BindThreshold Property
ChunkRequired Property
GetChunk Method
rdoColumn Object, **rdoColumns** Collection
rdoResultset Object, **rdoResultsets** Collection
Size Property
Type Property

_

ColumnSize Method (Remote Data) Applies To

rdoColumn Object

CreatePreparedStatement Method (Remote Data)

See Also Applies To

Creates a new **rdoPreparedStatement** object.

Syntax

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

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

Part	Description
prepstmt	An object expression that evaluates to the **rdoPreparedStatement** object you want to create.
connection	An object expression that represents the open **rdoConnection** object.
name	A **String** that is the name of the new **rdoPreparedStatement**. This part is required, but may be an empty string ("").
sqlstring	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

The **rdoPreparedStatement** corresponds to the ODBC prepared statement used to define a reusable SQL query that can contain parameters. You can execute the **rdoPreparedStatement** 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 **rdoPreparedStatement** objects than to use the **Execute** or **OpenResultset** method on objects other than the **rdoPreparedStatement**.

The **rdoPreparedStatement** is automatically appended to the **rdoPreparedStatements** collection.

If *name* is not provided, the **rdoPreparedStatement** is appended to the **rdoPreparedStatements** collection, and the **rdoPreparedStatement** can be used by referencing the *prepstmt* variable or the **rdoPreparedStatement** object's ordinal value.

If the object specified by *name* is already a member of the **rdoPreparedStatements** collection (including an empty string), a trappable error occurs. All **rdoPreparedStatement** objects are temporary – they are discarded when the **rdoConnection** object is closed.

To remove an **rdoPreparedStatement** object from an **rdoPreparedStatements** collection, use the **Close** method on the **rdoPreparedStatement**.

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

See Also

Close Method
Creating Parameter Queries
Execute Method
Name Property
OpenResultset Method
rdoConnection Object, **rdoConnections** Collection
rdoPreparedStatement Object, **rdoPreparedStatements** Collection
rdoResultset Object, **rdoResultsets** Collection
SQL Property
Type Property

CreatePreparedStatement Method (Remote Data) Applies To

rdoConnection Object

Delete Method (Remote Data)

See Also   Applies To

Deletes the current row in an updatable **rdoResultset** object.

Syntax

object.**Delete**

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

Remarks

Delete removes the current row and makes it inaccessible. The deleted row is removed from the **rdoResultset** cursor and the database. When you delete rows from an **rdoResultset**, there must be a current row in the **rdoResultset** before you use **Delete**; otherwise, a trappable error is triggered.

Once you delete a row in an **rdoResultset**, you must reposition the current row pointer to another row in the **rdoResultset** before performing an operation that accesses the current row. Although you can't edit or use the deleted row, it remains current until you reposition to another row. Once you move to another row, however, you can't make the deleted row current again.

When you position to a row that has been deleted by another user, or if you delete a common row in another **rdoResultset**, a trappable error occurs indicating that the row has been deleted. At this point, the current row is invalid and you must reposition to another valid row. For example, if you use a bookmark to position to a deleted row, a trappable error occurs.

You can undo a row deletion if you use transactions and the **RollbackTrans** method – assuming you use **BeginTrans** before using the **Delete** method.

Using **Delete** produces an error under any of the following conditions:

- There is no current row.
- The connection or **rdoResultset** is read-only.
- No columns in the row are updatable.
- The row has already been deleted.
- Another user has locked the data page containing your row.
- The user does not have permission to perform the operation.

See Also

AddNew Method
BeginTrans, **CommitTrans**, **RollbackTrans** Methods
Bookmark Property
LastModified Property
Name Property
rdoConnection Object, **rdoConnections** Collection
rdoEnvironment Object, **rdoEnvironments** Collection
rdoResultset Object, **rdoResultsets** Collection
rdoTable Object, **rdoTables** Collection
Refresh Method
Updatable Property

_

Delete Method (Remote Data) Applies To

rdoResultset Object

Edit Method (Remote Data)

See Also Applies To

Copies the current row from an updatable **rdoResultset** object to the copy buffer for subsequent editing.

Syntax

object.**Edit**

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

Remarks

Once you use the **Edit** method, changes made to the current row's columns are copied to the copy buffer. After you make the desired changes to the row, use the **Update** method to save your changes. The current row remains current after you use **Edit**.

Caution If you edit a row and then perform any operation that repositions the current row pointer to another row without first using **Update**, your changes to the edited row are lost without warning. In addition, if you close *object* or end the procedure which declares the result set or the parent **rdoConnection** object, your edited row might be discarded without warning.

When the **rdoResultset** object's **LockEdits** property setting is **True** (pessimistically locked) in a multiuser environment, the individual row or the data page containing the row remains locked from the time **Edit** is used until the updating is complete. If the **LockEdits** property setting is **False** (optimistically locked), the individual row or the data page containing the row is locked and the new row is compared with the pre-edited row just before it's updated in the database. If the row has changed since you last used the **Edit** method, the **Update** operation fails with a trappable error.

Note Not all data sources use page locking schemes to manage data concurrency. In some cases, data is locked on a row-by-row basis, therefore locks only affect the specific row being edited.

Using **Edit** produces an error under any of the following conditions:

- There is no current row.
- The connection or **rdoResultset** is read-only.
- No columns in the row are updatable.
- The **EditMode** property indicates that an **AddNew** or **Edit** is already in progress.
- Another user has locked the row or data page containing your row and the **LockEdits** property is **True**.

See Also

AddNew Method
BeginTrans, **CommitTrans**, **RollbackTrans** Methods
Delete Method
LockEdits Property
rdoResultset Object, **rdoResultsets** Collection
Update Method

Edit Method (Remote Data) Applies To

rdoResultset Object

Execute Method (Remote Data)

See Also Applies To

Runs an action query or executes an SQL statement that does not return rows on an object in the Applies To list.

Syntax

*connection***.Execute** *source*[, *options*]

*prepstmt***.Execute** [*options*]

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

Part	Description
connection	An object expression that evaluates to the **rdoConnection** object on which the query will run.
prepstmt	An object expression that evaluates to the **rdoPreparedStatement** object whose **SQL** property setting specifies the SQL statement to execute.
source	A string expression that contains the action query to execute or the name of an **rdoPreparedStatement**.
options	A **Variant** or constant that determines how the query is run, as specified in Settings.

Settings

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

Constant	Value	Description
rdAsyncEnable	32	Execute operation asynchronously.

Remarks

The **Execute** method is valid only for action queries. If you use **Execute** with a query that returns rows, a trappable error is generated and the result set is discarded. Because an action query doesn't return any rows, **Execute** doesn't return an **rdoResultset**. You can use the **Execute** method on queries that execute multiple statements, but none of these batched statements can return rows.

Use the **RowsAffected** property of the **rdoConnection** or **rdoPreparedStatement** object to determine the number of rows affected by the most recent **Execute** method. **RowsAffected** contains the number of rows deleted, updated, or inserted when executing an action query. When you use the **Execute** method to run an **rdoPreparedStatement**, the **RowsAffected** property of the **rdoPreparedStatement** object is set to the number of rows affected.

To execute the query asynchronously, use the **rdAsyncEnable** option. If set, the data source query processor immediately begins to process the query and returns to your application before the query is complete. Use the **StillExecuting** property to determine when the query processor is ready to return the results from the query. Use the **Cancel** method to terminate processing of an asynchronous query.

While it is possible to execute stored procedures using the **Execute** method, it is not recommended because the procedure's return value and output parameters are discarded and the procedure cannot return rows. Use the **rdoPreparedStatement** to execute stored procedures.

See Also

BeginTrans, **CommitTrans**, **RollbackTrans** Methods
Cancel Method
rdoConnection Object, **rdoConnections** Collection
rdoPreparedStatement Object, **rdoPreparedStatements** Collection
RowsAffected Property
SQL Property
StillExecuting Property

_

Execute Method (Remote Data) Applies To

rdoConnection Object
rdoPreparedStatement Object

GetChunk Method (Remote Data)

See Also Applies To

Returns all or a portion of the contents of an **rdoColumn** object with a data type of **rdTypeLONGVARBINARY** or **rdTypeLONGVARCHAR**.

Syntax

varname = *object* **!** *column***.GetChunk(***numbytes***)**

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

Part	Description
varname	The name of a **Variant** that receives the data from the **rdoColumn** object named by *column.*
object	An object expression that evaluates to an **rdoResultset** object containing the **rdoColumns** collection.
column	An object expression that evaluates to an **rdoColumn** object whose **ChunkRequired** property is **True**.
numbytes	A numeric expression that is the number of bytes you want to return.

Remarks

Chunk data columns are designed to store binary or text values that can range in size from a few characters to over 1.2GB and are stored in the database on successive data pages. In most cases, *chunk* data cannot be managed with a single operation so you must use the *chunk* methods to save and write data.   If the **ChunkRequired** property is **True** for a column, you must use the **GetChunk** and **AppendChunk** methods to manipulate column data.

If the **ChunkRequired** property is **True** for a column, you must use the **GetChunk** method to retrieve the data.   The **GetChunk** method moves a portion of the data from a *chunk* column to a variable.   The total number of bytes in the column is determined by executing the **ColumnSize** method.

The **GetChunk** method is used iteratively, copying column data to a variable, one segment or *chunk* at a time.   The chunk size is set by *numbytes*.   The starting point of the copy operation is initially 0, which causes data to be copied from the first byte of the column being read.   Subsequent calls to **GetChunk** get data from the first position after the previously read chunk.

Assign the bytes returned by **GetChunk** to *varname*. Due to memory requirements for the returned data and temporary storage, *numbytes* might be limited.   With 32-bit systems, this limitation is over 1.2GB, or more practically, the memory and disk capacity of your virtual memory system.

If *numbytes* is greater than the number of bytes in the column, the actual number of bytes in the column is returned.   After assigning the results of **GetChunk** to a **Variant** variable, you can use the **Len** function to determine the number of bytes returned.

Use the **AppendChunk** method to write successive blocks of data to the column and **GetChunk** to extract data from the column.   Certain operations (copying, for example) involve temporary strings.   If string space is limited, you may need to work with smaller segments of a *chunk* column instead of the entire column.

Use the **BindThreshold** property to specify the largest column size that will be automatically bound.

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.

See Also

AppendChunk Method
ColumnSize Method
ChunkRequired Property
rdoColumn Object, **rdoColumns** Collection
rdoResultset Object, **rdoResultsets** Collection
Type Property

GetChunk Method (Remote Data) Applies To

rdoColumn Object

GetRows Method (Remote Data)

See Also Applies To

Retrieves multiple rows of an **rdoResultset** into an array.

Syntax

array = *object*.**GetRows (***rows***)**

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

Part	Description
array	The name of a **Variant** type variable to store the returned data.
object	An object expression that evaluates to an object in the Applies To list.
rows	A **Long** value indicating the number of rows to retrieve.

Remarks

Use the **GetRows** method to copy one or more entire rows from an **rdoResultset** into a two-dimensional array. The first array subscript identifies the column and the second identifies the row number, as follows:

```
avarRows(intColumn)(intRow)
```

To get the first column value in the second row returned, use the following:

```
col1 = avarRows(0,1)
```

To get the second column value in the first row, use the following:

```
col2 = avarRows(1,0)
```

If more rows are requested than are available, only the available rows are returned. Use **Ubound** to determine how many rows are actually fetched, as the array is resized based on the number of rows returned. For example, if you return the results into a **Variant** called varA, you could determine how many rows were actually returned by using:

```
numReturned = Ubound(varA,2) + 1
```

The "+ 1" is used because the first data returned is in the 0th element of the array. The number of rows that can be fetched is constrained by available memory and should be chosen to suit your application−don't expect to use **GetRows** to bring your entire table or result set into an array if it is a large table.

GetRows does not return columns whose **ChunkRequired** property is **True**.

After a call to **GetRows**, the current row is positioned at the next unread row. That is, **GetRows** is equivalent to using the **Move** (*rows*) method.

If you are trying to fetch all the rows using multiple **GetRows** calls, use the **EOF** property to determine if there are rows available. **GetRows** returns less than the number requested either at the end of the **rdoResultset**, or if it cannot fetch a row in the range requested. For example, if a fifth row cannot be retrieved in a group of ten rows that you're trying to fetch, **GetRows** returns four rows and leaves currency on the row that caused the problem. It will not generate a run-time error.

See Also

ChunkRequired Property
Move Method
rdoResultset Object, **rdoResultsets** Collection

_

GetRows Method (Remote Data) Applies To

rdoResultset Object

Item Method (Remote Data)

See Also Applies To

Returns a specific member of a **collection,** either by position or by key.

Syntax

object.**Item(***index***)**

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

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

Remarks

If the value provided as *index* does not match any existing member of the collection, an error occurs.

The **Item** method is the default method for a collection. Therefore, the following lines of code are equivalent:

```
Print MyCollection(1)
Print MyCollection.Item(1)
```

See Also

Count Property

Value Property

Item Method (Remote Data) Applies To

rdoColumns Collection
rdoConnections Collection
rdoEnvironments Collection
rdoErrors Collection
rdoParameters Collection
rdoPreparedStatements Collection
rdoResultsets Collection

MoreResults Method (Remote Data)

See Also Applies To

Clears the current result set of any pending rows and returns a **Boolean** value that indicates if one or more additional result sets are pending.

Syntax

variable = *object***.MoreResults**

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

Part	Description
variable	A **Boolean** variable that indicates if additional result sets are found as described in Return Values.
object	An object expression that evaluates to an open **rdoResultset** object variable.

Return Values

The return values for *variable* are:

Value	Description
True	Additional result sets are ready to be processed.
False	All result sets in the **rdoResultset** have been processed.

Remarks

When the query used to create an **rdoResultset** returns more than one result set, use the **MoreResults** method to end processing of the current result set and test for subsequent result sets. If there are no additional result sets to process, the **MoreResults** method returns **False** and both **BOF** and **EOF** are set to **True**. In any case, using the **MoreResults** method flushes the current **rdoResultset**.

You can also use the **Cancel** method to flush the contents of an **rdoResultset**. However, **Cancel** also flushes any additional result sets not yet processed.

See Also

BOF, **EOF** Properties
Cancel Method
MoreResults Method
rdoResultset Object, **rdoResultsets** Collection

_

MoreResults Method (Remote Data) Applies To

rdoResultset Object

Move Method (Remote Data)

See Also Applies To

Repositions the current row in an **rdoResultset** object.

Syntax

object.**Move** *rows*[, *start*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
rows	A signed **Long** value that specifies the number of rows the position will move as described in Settings.
start	A **Variant** value that identifies a bookmark as described in Settings.

Settings

If *rows* is greater than 0, the position is moved forward (toward the end of the cursor). If *rows* is less than 0, the position is moved backward (toward the beginning of the cursor). If *rows* is equal to 0, any pending edits are discarded and the current row is refreshed from the data source.

If *start* is specified, the move begins relative to this bookmark. If *start* is not specified, **Move** begins from the current row.

Remarks

If using **Move** repositions the current row to a position before the first row, the position is moved to the beginning-of-file (**BOF**) position. If the **rdoResultset** contains no rows and its **BOF** property is set to **True**, using this method to move backward triggers a trappable run-time error. If either the **BOF** or **EOF** property is **True** and you attempt to use the **Move** method without a valid bookmark, a trappable error is triggered.

If using **Move** repositions the current row to a position after the last row, the position is moved to the end-of-file (**EOF**) position. If the **rdoResultset** contains no rows and its **EOF** property is set to **True**, then using this method to move forward produces a trappable run-time error.

If you use **Move** on an **rdoResultset** object based on an SQL-specific query or **rdoPreparedStatement**, the query is forced to completion and the **rdoResultset** object is fully populated.

If you use any method that repositions the current row pointer after using the **Edit** or **AddNew** method but before using the **Update** method, any changes made to the copy buffer are lost.

When you use **Move** on a forward-only **rdoResultset**, the *rows* argument must be a positive **Integer** and bookmarks aren't allowed, so you can only move forward − toward the end of the result set.

To make the first, last, next, or previous row in an **rdoResultset** the current row, use the **MoveFirst**, **MoveLast**, **MoveNext**, or **MovePrevious** method. To position the current row pointer based on an absolute row number, use the **AbsolutePosition** property. To position the current row pointer based on a percentage of the accessed rows of a result set, use the **PercentPosition** property.

See Also

AbsolutePosition Property
BOF, **EOF** Properties
Bookmark Property
MoveFirst, **MoveLast**, **MoveNext**, **MovePrevious** Methods
PercentPosition Property
rdoResultset Object, **rdoResultsets** Collection

_

Move Method (Remote Data) Applies To

rdoResultset Object

MoveFirst, MoveLast, MoveNext, MovePrevious Methods (Remote Data)

See Also Applies To

Repositions the current row pointer to the first, last, next, or previous row in a specified **rdoResultset** object and makes that row the current row.

Syntax

object.{**MoveFirst** | **MoveLast** | **MoveNext** | **MovePrevious**}

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

Remarks

Use the *Move* methods to reposition the current row pointer from row to row without applying a condition.

Caution If you edit the current row, be sure to save the changes using the **Update** method before you move to another row. If you move to another row without updating, your changes are lost without warning.

When you open the result set named by *object*, the first row is current and the **BOF** property is set to **False**. If the result set contains no rows, the **BOF** property is set to **True**, and there is no current row.

If the first or last row is already current when you use **MoveFirst** or **MoveLast**, the current row doesn't change.

If you use **MovePrevious** when the first row is current, the **BOF** property is set to **True**, and there is no current row. If you use **MovePrevious** again, an error occurs; **BOF** remains **True**.

If you use **MoveNext** when the last row is current, the **EOF** property is set to **True**, and there is no current row. If you use **MoveNext** again, an error occurs; **EOF** remains **True**.

If you use **MoveLast** on an **rdoResultset** object based on an SQL-specific query or **rdoPreparedStatement**, the query is forced to completion and the **rdoResultset** object is fully populated.

If you use any method that repositions the current row pointer after using the **Edit** or **AddNew** method but before using the **Update** method, any changes made to the copy buffer are lost.

You can't use the **MoveFirst** or **MovePrevious** method with a forward-only–type **rdoResultset**.

To move the position of the current row in an **rdoResultset** object a specific number of rows forward or backward, use the **Move** method.

To position the current row pointer based on an absolute row number, use the **AbsolutePosition** property. To position the current row pointer based on a percentage of the accessed rows of a result set, use the **PercentPosition** property.

See Also

AbsolutePosition Property
BOF, **EOF** Properties
Move Method
PercentPosition Property
rdoPreparedStatement Object, **rdoPreparedStatements** Collection
RowCount Property
rdoResultset Object, **rdoResultsets** Collection
Update Method

MoveFirst, MoveLast, MoveNext, MovePrevious Methods (Remote Data) Apply To

rdoResultset Object

OpenConnection Method (Remote Data)

See Also Applies To

Opens a connection to an ODBC data source and returns a reference to the **rdoConnection** object that represents a specific database.

Syntax

Set *connection* = *environment***.OpenConnection(***dsName*[, *prompt*[, *readonly*[, *connect*]]]**)**

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

Part	Description
connection	An object expression that evaluates to an **rdoConnection** object that you're opening.
environment	An object expression that evaluates to an existing **rdoEnvironment** object. You must provide an **rdoEnvironment** object.
dsName	A string expression that is the name of a registered ODBC data source name. If *dsName* doesn't refer to a valid ODBC data source name and the Data Source Name (DSN) parameter does not appear in the *connect* argument, or if it's a zero-length string (""), an error occurs if *prompt* is **rdDriverNoPrompt**; otherwise, the user is prompted to select from a list of registered data source names.
prompt	Based on this value, the ODBC driver manager prompts the user for connection information such as DSN, user name, and password. Use one of the following **Integer** constants that defines how the user should be prompted:

– **rdDriverNoPrompt**
–(Default)The driver manager uses the connection string provided in *dsName* and *connect*. If sufficient information is not provided, the **OpenConnection** method returns a trappable error.
– **rdDriverPrompt**
–The driver manager displays the ODBC Data Sources dialog box. The connection string used to establish the connection is constructed from the DSN selected and completed by the user via the dialog boxes, or, if no DSN is chosen, and the **DataSourceName** property is empty (in the case of the **RemoteData** control), the default DSN is used.
– **rdDriverComplete**
–If the connection string provided includes the DSN keyword, the driver manager uses the string as provided in *connect*. Otherwise it behaves as it does when **rdDriverPrompt** is specified.
– **rdDriverCompleteRequired**
–Behaves like **rdDriverComplete** except the driver disables the controls for any information not required to complete the connection. If the controls are disabled, users cannot select or specify missing arguments.

readonly	A **Boolean** value that is **True** if the connection is to be opened for read-only access, and **False** if the connection is to be opened for read/write access. If you omit this argument, the connection is opened for read/write access.
connect	A string expression used for opening the database. This string constitutes the ODBC connect arguments, and the argument values of *connect* are dependent on the ODBC driver. See the **Connect** property for syntax. If the *connect* argument is an empty string (""), the user name and password are taken from the **rdoEnvironment** object's **UserName** and **Password** properties.

Remarks

The open **rdoConnection** is automatically added to the **rdoConnections** collection.

Establishing an **rdoConnection** may require that the user have permission to access the network, the specific data source server, and the chosen database on that server.   Failure to meet these qualifications might result in failure to connect.

If you do not specify a database either through the DATABASE parameter of the *connect* argument or through the data source entry, the database opened when you establish a connection is determined by the default database assigned to the user.   This database is assigned by the database administrator.   In some cases, you may be able to change the default database by executing an action query containing an SQL command such as the Transact SQL USE *database* statement.

Note   The *connect* part of the **OpenConnection** method is coded differently than the *source* part of the **OpenDatabase** method as used with DAO. The *connect* part neither requires nor supports use of the "ODBC;" keyword at the beginning of the connect string. In addition, the *connect* part does not support use of the LOGINTIMEOUT argument.

Use the **Close** method on the object to close a database associated with an **rdoConnection**, remove the connection from the **rdoConnections** collection, and disconnect from the data source.

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

See Also

Close Method
Connect Property
rdoConnection Object, **rdoConnections** Collection
rdoEnvironment Object, **rdoEnvironments** Collection
UserName Property

_

OpenConnection Method (Remote Data) Applies To

rdoEnvironment Object

OpenResultset Method (Remote Data)

See Also Applies To

Creates a new **rdoResultset** object.

Syntax

Set *variable* = *connection*.**OpenResultset(***source*[, *type*[,*locktype*[, *options*]]]**)**

Set *variable* = *object*.**OpenResultset(**[*type*[,*locktype* [, *options*]]]**)**

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

Part	Description
variable	An object expression that evaluates to an **rdoResultset** object.
connection	An object expression that evaluates to an existing **rdoConnection** object you want to use to create the new **rdoResultset**.
object	An object expression that evaluates to an existing **rdoPreparedStatement** or **rdoTable** object you want to use to create the new **rdoResultset**. If *object* refers to an **rdoTable** object, the type of the new object is an **rdoResultset**. If *object* is not the name of an existing **rdoPreparedStatement**, or **rdoTable**, it is assumed to be an SQL query.
source	A **String** that specifies the source of the rows for the new **rdoResultset**. The source can be the name of an **rdoTable** object, the name of an **rdoPreparedStatement**, or an SQL statement that might returns rows.
type	An **Integer** or constant that specifies the type of cursor to create. If you don't specify a type, **OpenResultset** creates a forward-only **rdoResultset**. Use one of the following constants that defines the cursor type of the new **rdoResultset** object:

– **rdOpenForwardOnly**
–opens a forward-only
–type **rdoResultset** object. (Default)
– **rdOpenStatic**
–opens a static-type **rdoResultset** object.
– **rdOpenKeyset**
–opens a keyset-type **rdoResultset** object.
– **rdOpenDynamic**
–opens a dynamic-type **rdoResultset** object.
locktype An **Integer** that specifies the type of concurrency control. If you don't specify a *locktype*, **rdConcurReadOnly** is assumed. Use one of the following **Integer** constants that defines the *locktype* of the new **rdoResultset** object:
– **rdConcurLock**
–Pessimistic concurrency.
– **rdConcurReadOnly**
–Read-only (Default).
– **rdConcurRowver**
–Optimistic concurrency based on row ID.
– **rdConcurValues**
–Optimistic concurrency based on row values.

options	An **Integer** or constant that specifies characteristics of the new **rdoResultset**. Use the following constant:

– **rdAsyncEnable**
– Execute asynchronously.

Remarks

The new **rdoResultset** is automatically appended to the **rdoResultsets** collection.

If you use the **OpenResultset** method against an **rdoConnection**, include the *source* argument that specifies how the data rows are to be derived. For example, the *source* argument might contain an SQL query or the name of an **rdoTable** object.

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

If you use the **rdAsyncEnable** option, control returns to your application as soon as the query is begun, but before a result set is available. To test for completion of the query, use the **StillExecuting** property. The **rdoResultset** object is not valid until **StillExecuting** returns **False**.

If a new **rdoResultset** contains rows, the **BOF** and **EOF** properties are both **False** and the current row pointer is positioned at the beginning of the first result set. You can use the *Move* methods, or the **AbsolutePosition** or **PercentPosition** properties to reposition the current row pointer. For some data source and cursor combinations, you can also determine the number of rows available or processed by examining the **RowCount** property. However, use of this property might force the query to be run to completion – this can be a time-consuming operation for some data sources. The **RowCount** property returns -1 if it is not available.

If a new **rdoResultset** is the result of an action query or contains no rows, the **BOF** and **EOF** properties are both **True**.

To cancel the current result set and test for subsequent result sets, use the **MoreResults** method. To cancel all pending queries, use the **Cancel** method.

When you use the **Close** method against an **rdoResultset**, all pending queries are flushed and the **rdoResultset** is automatically dropped from the **rdoResultsets** collection.

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 **rdoPreparedStatement** 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 last result set 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.

Note Not all types of cursors and concurrency are supported by every ODBC data source driver. See **rdoResultset** for more information.

See Also

AbsolutePosition Property
BOF, **EOF** Properties
Cancel Method
Close Method
Connect Property
LockType Property
MoreResults Method
PercentPosition Property
rdoResultset Object, **rdoResultsets** Collection
ResultsetType Property
StillExecuting Property
Type Property
Understanding Cursors

_

OpenResultset Method (Remote Data) Applies To

rdoConnection Object
rdoPreparedStatement Object
rdoTable Object

rdoCreateEnvironment Method (Remote Data)

See Also Applies To

Creates a new **rdoEnvironment** object.

Syntax

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

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

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

Remarks

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

Unlike the other methods you use to create Remote Data Objects, **rdoCreateEnvironment** requires that you provide all of its parts.   In addition, **rdoEnvironment** objects aren't permanent and can't be saved.   Once you create an **rdoEnvironment** object, you can only modify the **UserName** and **Timeout** property settings.

You don't have to append the new **rdoEnvironment** object to a collection before you can use it − it is automatically appended to the **rdoEnvironments** collection.

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

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

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

See Also

Close Method
Name Property
Password Property
rdoDefaultUser, **rdoDefaultPassword** Properties
rdoEngine Object
rdoEnvironment Object, **rdoEnvironments** Collection
UserName Property

rdoCreateEnvironment Method (Remote Data) Applies To

rdoEngine Object

rdoRegisterDataSource Method (Remote Data)

See Also   Applies To

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

Syntax

rdoRegisterDataSource *dsName*, *driver*, *silent*, *attributes*

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

Part	Description
dsName	A string expression that is the name used in the **OpenConnection** method that refers to a block of descriptive information about the data source. For example, if the data source is an ODBC remote database, it could be the name of the server.
driver	A string expression that is the name of the ODBC driver. This isn't the name of the ODBC driver dynamic link library (DLL) file. For example, *SQL Server* is a driver name, but *SQLSRVR.DLL* is the name of a DLL file. You must have ODBC and the appropriate driver already installed.
silent	A **Boolean** value that is **True** if you don't want to display the ODBC driver dialog boxes that prompt for driver-specific information, or **False** if you do want to display the ODBC driver dialog boxes. If *silent* is **True**, *attributes* must contain all the necessary driver-specific information or the dialog boxes are displayed anyway.
attributes	A string expression that is a list of keywords to be added to the ODBC.INI file. The keywords are in a carriage-return-delimited string.

Remarks

The ODBC driver needs connection information when the database engine opens the data source during a session.

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

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

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

See Also

rdoConnection Object, **rdoConnections** Collection
rdoEngine Object
OpenConnection Method

_

rdoRegisterDataSource Method (Remote Data) Applies To

rdoEngine Object

Refresh Method (Remote Data)

See Also Applies To

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

Syntax

object.**Refresh**

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

Remarks

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

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

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

If both the **BOF** and **EOF** property settings of the **rdoResultset** object are **True**, or the **RowCount** property is set to 0 after you use the **Refresh** method, the query didn't return any rows and the new **rdoResultset** contains no data.

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

See Also

BOF, **EOF** Properties
Close Method
Execute Method
rdoTables Collection
RemoteData Control
Requery Method
RowCount Property

_

Refresh Method (Remote Data) Applies To

rdoTables Collection
rdoColumns Collection
RemoteData Control

Requery Method (Remote Data)

See Also Applies To

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

Syntax

object.**Requery**

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

Remarks

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

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

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

If both the **BOF** and **EOF** property settings of the **rdoResultset** object are **True**, or the **RowCount** property is set to 0 after you use the **Requery** method, the query didn't return any rows and the **rdoResultset** contains no data.

You can't use the **Requery** method on **rdoResultset** objects whose **Restartable** property is set to **False**.

See Also

BOF, **EOF** Properties
Execute Method
rdoParameter Object, **rdoParameters** Collection
rdoPreparedStatement Object, **rdoPreparedStatements** Collection
rdoResultset Object, **rdoResultsets** Collection
Restartable Property
RowCount Property

_

Requery Method (Remote Data) Applies To

rdoResultset Object

Update Method (Remote Data)

See Also Applies To

Saves the contents of the copy buffer row to a specified updatable **rdoResultset** object.

Syntax

object.**Update**

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

Remarks

Use **Update** to save the current row and any changes you've made to it to the underlying database table(s).

Changes to the current row are lost if:

- You use the **Edit** or **AddNew** method, and then reposition the current row pointer to another row without first using **Update**.
- You use **Edit** or **AddNew**, and then use **Edit** or **AddNew** again without first using **Update**.
- You cancel the update with the **CancelUpdate** method.
- You set the **Bookmark** property to another row.
- You close the result set referred to by *object* without first using **Update**.

To edit a row, use the **Edit** method to copy the contents of the current row to the copy buffer. If you don't use **AddNew** or **Edit** first, an error occurs when you use **Update** or attempt to add a new row.

To add a new row, use the **AddNew** method to initialize the copy buffer.

When the **rdoResultset** object's **LockEdits** property setting is **True** (pessimistically locked) in a multiuser environment, the data page or row remains locked from the time **Edit** is used until the **Update** method is executed. If the **LockEdits** property setting is **False** (optimistically locked), the data page or row is locked and compared with the pre-edited row just before it is updated in the database. If the row has changed since you used the **Edit** method, the **Update** operation fails with a trappable error. To revert to the row as the other user changed it, refresh the current row using the *Move* methods, or set the **Bookmark** property to itself. To add a new row to a result set, use the **AddNew** method.

Using **Update** produces an error under any of the following conditions:

- There is no current row.
- The connection or **rdoResultset** is read-only.
- No columns in the row are updatable.
- You do not have an **Edit** or **AddNew** operation pending.
- Another user has locked the row or data page containing your row.
- The user does not have permission to perform the operation.

See Also

AddNew Method
BeginTrans, **CommitTrans**, **RollbackTrans** Methods
Bookmark Property
Edit Method
LockEdits Property
OpenResultset Method
rdoResultset Object, **rdoResultsets** Collection
UpdateControls Method
UpdateRow Method

_

Update Method (Remote Data) Applies To

rdoResultset Object

UpdateControls Method (Remote Data)

See Also Applies To

Gets the current row from a **RemoteData** control's **rdoResultset** object and displays the appropriate data in controls bound to a **RemoteData** control.

Syntax

object.**UpdateControls**

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

Remarks

Use this method to restore the contents of bound controls to their original values, as when a user makes changes to data and then decides to cancel the changes.

This method creates the same effect as making the current row current again, except that no events occur. By not invoking any events, this method can be used to simplify an update operation because no additional validation or change event procedures are triggered.

See Also

Edit Method
Update Method
UpdateRow Method

_

UpdateControls Method (Remote Data) Applies To

RemoteData Control

UpdateRow Method (Remote Data)

See Also Applies To

Saves the current values of bound controls to the database.

Syntax

object.**UpdateRow**

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

Remarks

Use this method to save the current contents of bound controls to the database during the Validate event, but without triggering the Validate event again. You can use this method to avoid triggering the Validate event. Using this method avoids a cascading event.

The **UpdateRow** method has the same effect as executing the **Edit** method, changing a column, and then executing the **Update** method, except that no events occur.

Whenever you attempt to update a row in the database, any validation rules must be satisfied before the row is written to the database. In the case of Microsoft SQL Server, these rules are established by Transact SQL defaults, rules, and triggers written to enforce referential and data integrity.

An update may not occur because of any of the following reasons, which can also trigger a trappable error:

- The page containing the row or the row itself is locked.
- The database or **rdoResultset** object isn't updatable.
- The user doesn't have permission to perform the operation.

See Also

AddNew Method
BeginTrans, **CommitTrans**, **RollbackTrans** Methods
Bookmark Property
Edit Method
LockEdits Property
OpenResultset Method
rdoResultset Object, **rdoResultsets** Collection
RemoteData Control
UpdateControls Method

UpdateRow Method (Remote Data) Applies To

RemoteData Control

Clear Method (Remote Data)

See Also 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.

See Also

rdoError Object, **rdoErrors** Collection

_

Clear Method (Remote Data) Applies To

rdoErrors Collection

Move Method (RemoteData Control)

Moves an **MDIForm**, **Form**, or control. Doesn't support named arguments.

Syntax

*object***.Move** *left*, *top*, *width*, *height*

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

Part	Description
object	Optional. An object expression that evaluates to a **RemoteData** control. If *object* is omitted, the form with the focus is assumed to be *object*.
left	Required. Single-precision value indicating the horizontal coordinate (x-axis) for the left edge of *object*.
top	Optional. Single-precision value indicating the vertical coordinate (y-axis) for the top edge of *object*.
width	Optional. Single-precision value indicating the new width of *object*.
height	Optional. Single-precision value indicating the new height of *object*.

Remarks

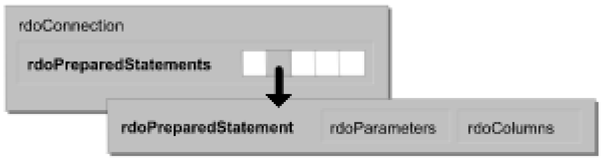
Only the *left* argument is required. However, to specify any other arguments, you must specify all arguments that appear in the syntax before the argument you want to specify. For example, you can't specify *width* without specifying *left* and *top*. Any trailing arguments that are unspecified remain unchanged.

For forms and controls in a **Frame** control, the coordinate system is always in twips. Moving a form on the screen or moving a control in a **Frame** is always relative to the origin (0,0), which is the upper-left corner. When moving a control on a **Form** object or in a **PictureBox** (or an MDI child form on an **MDIForm** object), the coordinate system of the container object is used. The coordinate system or unit of measure is set with the **ScaleMode** property at design time. You can change the coordinate system at run time with the **Scale** method.

rdoConnection Object, rdoConnections Collection

See Also Properties Methods Summary

– An **rdoConnection** object represents an open connection to a remote data source and a specific database on that data source.
– An **rdoConnections** collection contains all open **rdoConnection** objects opened or created in an **rdoEnvironment** object of the remote database engine.



Remarks

You can open connections to remote ODBC data sources and create **rdoConnection** objects with either the **RemoteData** control or the **OpenConnection** method of an **rdoEnvironment** object.

Once a connection is established, you can manipulate a database associated with the **rdoConnection** using the **rdoConnection** object and its methods and properties. The database that is opened is determined by either the default database assigned to the user name, or by a specific default database specified with the DATABASE argument used when the **rdoConnection** is created. Some SQL dialects, such as Transact SQL for SQL Server, support SQL syntax you can use to specify a particular database for a query.

You can manipulate databases with the methods and properties of the **rdoConnection** object. For example, you can:

– Use the **Execute** method to run an action query or pass an SQL statement to a database for execution.
– Use the **OpenResultset** method to create a new **rdoResultset** object.
– Use the **CreatePreparedStatement** method to create a new **rdoPreparedStatement** object.
– Use the **RowsAffected** property to determine how many rows were affected by the last operation.
– Use the **Close** method to close an open connection, deallocate the connection handle, and terminate the connection. Any open **rdoResultset, rdoTable,** or **rdoPreparedStatement** objects are closed automatically when the **rdoConnection** object is closed. However, if the **rdoConnection** object simply loses scope, any open **rdoResultset**, **rdoTable** or **rdoPreparedStatement** objects remain open until the **rdoConnection** or the objects are explicitly closed.
– Use the **QueryTimeout** property to specify how long the ODBC driver manager should wait before abandoning a query.
– Use the **Transactions** property to determine if the connection supports transactions, which you can implement using the **BeginTrans**, **CommitTrans**, and **RollbackTrans** methods.
– Use the ODBC API with the **hDbc** property to set connection options.

The **Connect** property contains the *connect* argument used in the **OpenConnection** method, or the **Connect** property of the **RemoteData** control.

The **Name** property setting of an **rdoConnection** specifies the data source name (DSN) parameter used to open the connection. You can refer to any **rdoConnection** object by its **Name** property setting using this syntax:

rdoConnections("*name***")**

You can also refer to the object by its ordinal number using this syntax (which refers to the first member of the **rdoConnections** collection):

rdoConnections(0**)**

Note To use the Remote Data Objects (RDO), you must set a reference to the Microsoft Remote Data Object 1.0 object library in the Visual Basic References dialog box.

See Also

Close Method
Name Property
OpenConnection Method
QueryTimeout Property
rdoEnvironment Object, **rdoEnvironments** Collection
rdoPreparedStatement Object, **rdoPreparedStatements** Collection
rdoResultset Object, **rdoResultsets** Collection
rdoTable Object, **rdoTables** Collection
RemoteData Control

rdoConnection Object, rdoConnections Collection Summary

rdoConnection Object

The **rdoConnection** object contains these collections, methods, and properties.

Collections
rdoPreparedStatements
rdoResultsets
rdoTables

Methods
BeginTrans, **CommitTrans**, **RollbackTrans**
Cancel
Close
CreatePreparedStatement
Execute
OpenResultset

Properties
AsyncCheckInterval
Connect
hDbc
Name
QueryTimeout
RowsAffected
StillExecuting
Transactions
Updatable
Version

rdoConnections Collection

The **rdoConnections** collection contains one method and one property.

Method
Item

Property
Count

rdoEngine Object

See Also Properties Methods Summary

The **rdoEngine** object represents the remote data source. As the top-level object, it contains all other objects in the hierarchy of Remote Data Objects (RDO).

rdoEngine
rdoEnvironments

Remarks

The **rdoEngine** object can represent a remote database engine or another data source managed by the ODBC driver manager as a database. The **rdoEngine** object is a predefined object, therefore you can't create additional **rdoEngine** objects and it isn't an element of any collection.

Use the **rdoEngine** object to set data source parameters and the default **rdoEnvironment**, control the ODBC data source and manipulate its properties, and perform tasks on temporary objects that aren't members of collections. For example, you can:

- Use the **rdoEnvironments** collection to examine open **rdoEnvironment** objects.
- Use the **rdoErrors** collection to examine information about errors.
- Use the **rdoDefaultLoginTimeout** property to obtain or set the default value used for ODBC login timeout.
- Use the **rdoDefaultCursorDriver** property to determine if the ODBC driver manager creates local or server-side cursors.
- Use the **rdoDefaultUser** and **rdoDefaultPassword** properties to obtain or set the user name and password to be used when opening connections if no specific values are supplied.
- Use the **rdoDefaultErrorThreshold** property to obtain or set the default value used for error numbers that constitute a fatal error
− one that terminates a query and triggers a trappable Visual Basic error.
- Use the **rdoRegisterDataSource** method to create a new data source entry either in the Windows System Registry (32-bit) or in the ODBC.INI file (16-bit).
- Use the **rdoCreateEnvironment** method to create a new **rdoEnvironment** object.

Note To use the Remote Data Objects (RDO), you must set a reference to the Microsoft Remote Data Object 1.0 object library in the Visual Basic References dialog box.

See Also

rdoConnection Object, **rdoConnections** Collection
rdoCreateEnvironment Method
rdoDefaultCursorDriver Property
rdoDefaultErrorThreshold Property
rdoDefaultLoginTimeout Property
rdoDefaultUser, rdoDefaultPassword Properties
rdoError Object, **rdoErrors** Collection
rdoRegisterDataSource Method
Remote Data Objects and Collections

rdoEngine Object Summary

rdoEngine Object

The **rdoEngine** object contains these collections, methods, and properties.

Collections

rdoEnvironments
rdoErrors

Methods

rdoCreateEnvironment
rdoRegisterDataSource

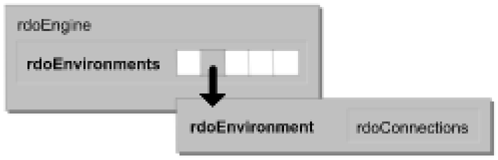
Properties

rdoDefaultCursorDriver
rdoDefaultErrorThreshold
rdoDefaultLoginTimeout
rdoDefaultPassword
rdoDefaultUser
rdoVersion

rdoEnvironment Object, rdoEnvironments Collection

See Also Properties Methods Summary

- An **rdoEnvironment** object defines a logical set of connections for a particular user name. It contains open databases, provides mechanisms for simultaneous transactions, and provides a security context for data manipulation language (DML) operations on the database.
- The **rdoEnvironments** collection contains all active **rdoEnvironment** objects of the **rdoEngine** object.



Remarks

An **rdoEnvironment** object corresponds to an ODBC environment that can be referred to by the **rdoEnvironment** object's **hEnv** property. All **rdoEnvironment** objects share a common **hEnv** value that is created on an application basis.

rdoEnvironment objects are created with the **rdoCreateEnvironment** method of the **rdoEngine** object. Newly created **rdoEnvironment** objects are automatically appended to the **rdoEnvironments** collection.

The user name and password information from the **rdoEnvironment** is used to establish the connection if these values are not supplied in the *connect* argument of the **OpenConnection** method, or in the **Connect** property of the **RemoteData** control.

The **rdoEnvironment** also determines transaction scope. Committing an **rdoEnvironment** transaction commits all open **rdoConnection** databases and their corresponding open **rdoResultset** objects. This does not imply a two-phase commit operation – simply that individual **rdoConnection** objects are instructed to commit any pending operations.

The default **rdoEnvironment** is created automatically when the **RemoteData** control is initialized, or the first remote data object is referenced in code. The **Name** property of **rdoEnvironments**(0) is "Default_Environment". The user name and password for **rdoEnvironments**(0) are both "".

Use the **rdoEnvironment** object to manage the current ODBC environment, or to start an additional connection. In an **rdoEnvironment**, you can open multiple databases, manage transactions, and establish security based on user names and passwords. For example, you can:

- Create an **rdoEnvironment** object using the **Name**, **Password**, and **UserName** properties to establish a named, password-protected environment. The environment creates a scope in which you can open multiple connections and conduct one instance of nested transactions.
- Use the **OpenConnection** method to open one or more existing connections in that **rdoEnvironment**.
- Use the **BeginTrans**, **CommitTrans**, and **RollbackTrans** methods to manage batched transaction processing within an **rdoEnvironment** across several connections.
- Use several **rdoEnvironment** objects to conduct multiple, simultaneous, independent, and overlapping transactions.
- Use the **Close** method to terminate an environment and the connection.

When you use transactions, all databases in the specified **rdoEnvironment** are affected– even if multiple **rdoConnection** objects are opened in the **rdoEnvironment**. For example, if you use a **BeginTrans** method against one of the databases visible from the connection, update several rows in the database, and then delete rows in another **rdoConnection** database. When you use the **RollbackTrans** method, both the update and delete operations are rolled back. You can create additional **rdoEnvironment** objects to manage transactions independently across **rdoConnection** objects. Note that transactions executed by multiple **rdoEnvironment** objects are serialized and are not atomic operations. Because of this, their success or failure is not interdependent. This is an example of batched transactions.

You can execute nested transactions *only* if your data source supports them. For example, on a single connection, you can execute a BEGIN TRANS SQL statement, execute several UPDATE queries, and another BEGIN TRANS statement. Any operations executed after the second BEGIN TRANS SQL statement can be rolled back independently of the statements executed after the first BEGIN TRANS. This is an example of nested transactions. To commit the first set of UPDATE statements, you must execute a COMMIT TRANS statement, or a ROLLBACK TRANS statement for each BEGIN TRANS executed.

The **Name** property of **rdoEnvironment** objects is set from the *name* argument passed to the **rdoCreateEnvironment** method. You can refer to any other **rdoEnvironment** object by specifying its **Name** property setting using this syntax:

rdoEnvironments("*name*")

or simply:

rdoEnvironments!*name*

You can also refer to **rdoEnvironment** objects by their position in the **rdoEnvironments** collection using this syntax (where *n* is the *n*th member of the zero-based **rdoEnvironments** collection):

rdoEngine.rdoEnvironments(*n*)

or simply:

rdoEnvironments(*n*)

Note To use the Remote Data Objects (RDO), you must set a reference to the Microsoft Remote Data Object 1.0 object library in the Visual Basic References dialog box.

See Also

Connect Property
hEnv Property
rdoCreateEnvironment Method
rdoConnection Object, **rdoConnections** Collection
rdoEngine Object
RemoteData Control

rdoEnvironment Object, rdoEnvironments Collection Summary

rdoEnvironment Object

The **rdoEnvironment** object contains these collections, methods, and properties.

Collections

rdoConnections

Methods

BeginTrans, **CommitTrans**, **RollbackTrans**
Close
OpenConnection

Properties

CursorDriver
hEnv
LoginTimeout
Name
Password
UserName

rdoEnvironments Collection

The **rdoEnvironments** contains one method and one property.

Method

Item

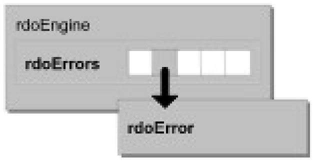
Property

Count

rdoError Object, rdoErrors Collection

See Also Properties Methods Summary

- **rdoError** object
- contains details about remote data access errors.
- **rdoErrors** collection
- contains all stored **rdoError** objects, which pertain to a single operation involving Remote Data Objects (RDO).



Remarks

Any operation involving remote data objects can generate one or more errors. As each error occurs, one or more **rdoError** objects are placed in the **rdoErrors** collection of the **rdoEngine** object. When another RDO operation generates an error, the **rdoErrors** collection is cleared, and the new set of **rdoError** objects is placed in the **rdoErrors** collection. RDO operations that don't generate an error have no effect on the **rdoErrors** collection.

If the severity of the error number is below the error threshold as specified in either the **rdoDefaultErrorThreshold** or **ErrorThreshold** property, then a trappable error is triggered when the error is detected. Otherwise, an **rdoError** object is simply appended to the **rdoErrors** collection.

Use the **rdoError** object to determine the type and severity of any errors generated by the **RemoteData** control or RDO operations. For example, you can:

- Use the **Description** property to display a text message describing the error.
- Use the **Number** property to determine the native data source error number.
- Use the **Source** property to determine the source of the error and the object class causing the error.
- Use the **SQLRetCode** and **SQLState** properties to determine the ODBC return code and **SQLState** flags.
- Use the **Clear** method on the **rdoErrors** collection to remove all **rdoError** objects.

In most cases, it is not necessary to use the **Clear** method because the **rdoErrors** collection is cleared automatically when a new error occurs.

Members of the **rdoErrors** collection aren't appended as is typical with other collections. The most general errors are placed at the end of the collection (**Count** -1), and the most detailed errors are placed at index 0. Because of this implementation, you can determine the root cause of the failure by examining **rdoErrors**(0).

The set of **rdoError** objects in the **rdoErrors** collection describes one error. The first **rdoError** object is the lowest level error, the second is the next higher level, and so forth. For example, if an ODBC error occurs while the **RemoteData** control tries to create an **rdoResultset** object, the last **rdoError** object contains the RDO error indicating the object couldn't be opened. The first error object contains the lowest level ODBC error. Subsequent errors contain the ODBC errors returned by the various layers of ODBC. In this case, the driver manager, and possibly the driver itself, returns separate errors which generate **rdoError** objects.

Note To use the Remote Data Objects (RDO), you must set a reference to the Microsoft Remote Data Object 1.0 object library in the Visual Basic References dialog box.

See Also

RemoteData Control
rdoEngine Object
rdoResultset Object, **rdoResultsets** Collection

rdoError Object, rdoErrors Collection Summary

rdoError Object

An **rdoError** object contains these properties.

Properties	
Description	**Source**
HelpContext	**SQLRetCode**
HelpFile	**SQLState**
Number	

rdoErrors Collection

The **rdoErrors** collection contains these methods and properties.

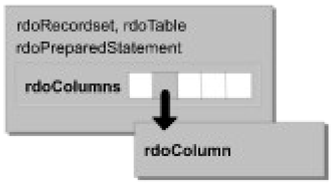
Method
Clear
Item

Property
Count

rdoColumn Object, rdoColumns Collection

See Also Properties Methods Summary

- An **rdoColumn** object represents a column of data with a common data type and a common set of properties.
- An **rdoColumns** collection contains all **rdoColumn** objects of an **rdoResultset**, **rdoPreparedStatement**, or **rdoTable** object.



Remarks

The **rdoTable** object's **rdoColumns** collection contains specifications for the data columns. You use the **rdoColumn** object in an **rdoTable** to map a base table's column structure. However, you cannot alter the structure of a database table using RDO properties and methods.

The **rdoTable**, **rdoResultset**, or **rdoPreparedStatement** object's **rdoColumns** collection represents the **rdoColumn** objects in a row of data. You use the **rdoColumn** object in an **rdoResultset** or **rdoPreparedStatement** to read and set values for the data columns in the current row of the object.

An **rdoColumn** object's name is determined by the name used to define the column in the data source table or by the name assigned to it in an SQL query.

You manipulate columns using an **rdoColumn** object and its methods and properties. For example, you can:

- Use the **SourceColumn** and **SourceTable** property settings to locate the original source of the data.
- Use the **Type** and **Size** property settings to get the data type and size of the data.
- Use the **OrdinalPosition** property to get presentation order of the **rdoColumn** objects in an **rdoColumns** collection.
- Use the **Attributes** and **Required** property settings to determine optional characteristics and if **Null**s are permitted in the column.
- Use the **AllowZeroLength** property setting to get the zero-length string handling setting.
- Use the **Updatable** property to see if the column can be changed.
- Use the **AppendChunk, ColumnSize**, and **GetChunk** methods to manipulate columns that require the use of these methods, as determined by the **ChunkRequired** property.
- Use the **Value** property of an **rdoColumn** to extract data from a specified column.

You can refer to the **Value** property of an **rdoColumn** object by:

- Referencing the **Name** property setting using this syntax:

  rdoColumns("***name***")

  -Or-

  rdoColumns!*name*
- Referencing its ordinal position in the **rdoColumns** collection using this syntax:

  rdoColumns(0**)**

When the **rdoColumn** object is accessed as part of an **rdoResultset** object, data from the current row is visible in the **rdoColumn** object's **Value** property. To manipulate data in the **rdoResultset**, you don't usually reference the **rdoColumns** collection directly. Instead, use syntax that references the **rdoColumns** collection as the default collection of the **rdoResultset**.

Note To use the Remote Data Objects (RDO), you must set a reference to the Microsoft Remote Data Object 1.0 object library in the Visual Basic References dialog box.

See Also

rdoPreparedStatement Object, **rdoPreparedStatements** Collection
rdoResultset Object, **rdoResultsets** Collection
rdoTable Object, **rdoTables** Collection

rdoColumn Object, rdoColumns Collection Summary

rdoColumn Object

An **rdoColumn** object contains these properties and methods.

Properties

AllowZeroLength
Attributes
ChunkRequired
Name
OrdinalPosition
Required
Size
SourceColumn
SourceTable
Type
Updatable
Value

Methods

AppendChunk
ColumnSize
GetChunk

rdoColumns Collection

The **rdoColumns** collection contains one method and one property.

Method

Item

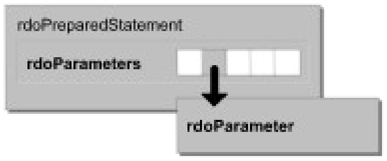
Property

Count

rdoParameter Object, rdoParameters Collection

See Also Properties Summary

- An **rdoParameter** object represents a parameter associated with an **rdoPreparedStatement** object.
- An **rdoParameters** collection contains all the **rdoParameter** objects of an **rdoPreparedStatement** object.



Remarks

Using the properties of an **rdoParameter** object, you can set a query parameter that can be changed before the query is run. You can:

- Use the **Direction** property setting to determine if the parameter is an input, output, or input/output parameter, or a return value.
- Use the **Type** property setting to determine the data type of the **rdoParameter**. Data types are identical to those specified by the **rdoColumn.Type** property.
- Use the **Value** property (the default property of an **rdoParameter**) to pass values to the SQL queries containing parameter markers used in **rdoPreparedStatement.Execute** or **rdoPreparedStatement.OpenResultset** methods. For example:

```
MyPreparedStatement.rdoParameters(0) = 5
```

The **rdoParameters** collection provides information only about existing parameters. You can't append objects to or delete objects from the **rdoParameters** collection.

Members of the **rdoParameters** collection are named "Param*n*" where *n* is the **rdoParameter** object's ordinal number. For example, if an **rdoParameters** collection has two members, they are named "Param0" and "Param1".

You can refer to the **Value** property of an **rdoParameter** object by:

- Referencing the **Name** property setting using this syntax:
  rdoParameters("*name***")**

  -Or-

  rdoParameters!*name*
- Referencing its ordinal position in the **rdoParameters** collection using this syntax:
  rdoParameters(0**)**

Note To use the Remote Data Objects (RDO), you must set a reference to the Microsoft Remote Data Object 1.0 object library in the Visual Basic References dialog box.

See Also

Direction Property
Execute Method
OpenResultset Method
rdoPreparedStatement Object, **rdoPreparedStatements** Collection
rdoResultset Object, **rdoResultsets** Collection
Type Property

rdoParameter Object, rdoParameters Collection Summary

rdoParameter Object

An **rdoParameter** object represents a parameter associated with an **rdoPreparedStatement** object created from a parameter query.

Properties

Direction
Name
Type
Value

rdoParameters Collection

An **rdoParameters** collection contains one method and one property.

Method

Item

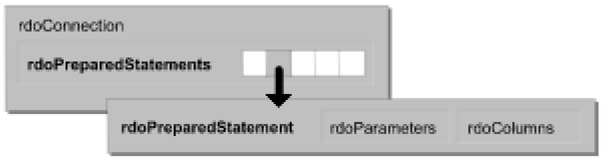
Property

Count

rdoPreparedStatement Object, rdoPreparedStatements Collection

See Also Properties Methods Summary

- An **rdoPreparedStatement** object is a prepared query definition.
- An **rdoPreparedStatements** collection contains all **rdoPreparedStatement** objects in an **rdoConnection**.



Remarks

An **rdoPreparedStatement** is like a compiled SQL statement. You can use the properties of an **rdoPreparedStatement** object to define a query. For example, you can:

- Set query parameters using the **rdoPreparedStatement** object's **rdoParameters** collection.
- Use the **SQL** property setting, set its parameters, and then run the query.
- Use the **Type** property to determine whether the query selects rows from an existing table (select query), performs an action (an action query), or is a procedure.
- Use the **RowsetSize** property setting to determine how many rows are buffered internally when building a cursor.
- Use the **QueryTimeout** property to indicate how long the driver manager waits before abandoning a query.
- Use the **BindThreshold** property to indicate the largest column to be automatically bound.
- Use the **ErrorThreshold** property to indicate the error level that constitutes a trappable error.
- Use the **KeysetSize** property to indicate the size of the keyset buffer when creating cursors.
- Use the **MaxRows** property to indicate the maximum number of rows to be returned by a query.
- Use the **RowsAffected** property to indicate how many rows are affected by an action query.
- Use the **Updatable** property to see if the result set generated by an **rdoPreparedStatement** can be updated.
- Use the **Close** method to terminate an **rdoPreparedStatement** query and release its resources.
- Use the **OpenResultset** method to create an **rdoResultset** based on the **OpenResultset** arguments and properties of the **rdoPreparedStatement**.
- Use the **Execute** method to run an action query using **SQL** and other **rdoPreparedStatement** properties, including any values specified in the **rdoParameters** collection.
- Use the **LogMessages** property to activate ODBC tracing.

rdoPreparedStatement objects are the preferred way to use the native SQL dialect of an external database engine. For example, you can create a Transact SQL query (as used on Microsoft SQL Server) and store it in an **rdoPreparedStatement** object.

You refer to an **rdoPreparedStatement** object by its **Name** property setting using this syntax:

rdoPreparedStatements("*name***")**

or

rdoPreparedStatements!*name*

You can also refer to **rdoPreparedStatement** objects by their position in the

rdoPreparedStatements collection using this syntax (where *n* is the *n*th member of the zero-based **rdoPreparedStatements** collection):

rdoPreparedStatements(*n*)

Note To use the Remote Data Objects (RDO), you must set a reference to the Microsoft Remote Data Object 1.0 object library in the Visual Basic References dialog box.

See Also

CreatePreparedStatement Method
OpenResultset Method
rdoColumn Object, **rdoColumns** Collection
rdoConnection Object, **rdoConnections** Collection
rdoParameter Object, **rdoParameters** Collection
Understanding Cursors

rdoPreparedStatement Object, rdoPreparedStatements Collection Summary

rdoPreparedStatement Object

The **rdoPreparedStatement** object contains these collections, methods, and properties.

Collections

rdoColumns
rdoParameters (default)

Methods

Cancel
Close
Execute
OpenResultset

Properties

BindThreshold
Connect
ErrorThreshold
hStmt
KeysetSize
LockType
LogMessages
MaxRows
Name
QueryTimeout
RowsAffected
RowsetSize
SQL
StillExecuting
Type
Updatable

rdoPreparedStatements Collection

The **rdoPreparedStatements** collection contains one method and one property.

Method

Item

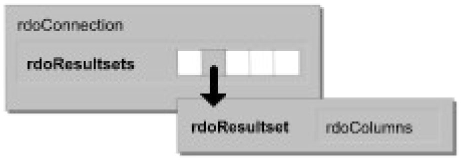
Property

Count

rdoResultset Object, rdoResultsets Collection

See Also Properties Methods Summary

– An **rdoResultset** object represents the rows that result from running a query.
– The **rdoResultsets** collection contains all open **rdoResultset** objects in an **rdoConnection**.



Remarks

When you use remote data objects, you interact with data almost entirely using **rdoResultset** objects. **rdoResultset** objects are created using the **RemoteData** control, or the **OpenResultset** method of the **rdoPreparedStatement**, **rdoTable,** or **rdoConnection** object.

When you execute a query that contains one or more SQL SELECT statements, the data source returns zero or more rows in an **rdoResultset** object. All **rdoResultset** objects are constructed using rows and columns.

A single **rdoResultset** can contain zero or any number of result sets – so-called "multiple" result sets. Once you have completed processing the first result set in an **rdoResultset** object, use the **MoreResults** method to discard the current **rdoResultset** rows and activate the next **rdoResultset**. You can process individual rows of the new result set just as you processed the first **rdoResultset**. You can repeat this until the **MoreResults** method returns **False**.

A new **rdoResultset** is automatically added to the **rdoResultsets** collection when you open the object, and it's automatically removed when you close it. Only one **rdoResultset** object is active at any one time.

Each of the **rdoResultset** rows must be processed before you can process subsequent result sets. To process result set rows, use the *Move* methods to position to individual rows, or the **MoveLast** method to position to the last row of the **rdoResultset**. Using the **MoreResults** method discards the current **rdoResultset** rows and activates the next **rdoResultset**.

You can choose the type of **rdoResultset** object you want to create using the *type* argument of the **OpenResultset** method. If you don't specify a *type*, the **RemoteData** control creates a keyset-type **rdoResultset**. There are four types of **rdoResultset** objects based on the type of cursor that is created to access the data:

– Forward-only
–type **rdoResultset**
–individual rows in the result set can be accessed and can be updatable (when using server-side cursors), but the current row pointer can only be moved toward the end of the **rdoResultset**. A forward-only
–type **rdoResultset** can contain columns from one or more tables in a database. Forward-only result sets are not cursors.
– Static-type **rdoResultset**
–a static copy of a set of rows that you can use to find data or generate reports. A static-type **rdoResultset** can contain columns from one or more tables in a database. Static cursors might be updatable when using either the ODBC cursor library or server-side cursors, depending on which drivers are supported and whether the source data can be updated.
– Keyset-type **rdoResultset**
–the result of a query that can have updatable rows. Movement within the keyset is unrestricted. A keyset-type **rdoResultset** is a dynamic set of rows that you can use to add, change, or delete rows from an underlying database table or tables. A keyset-type **rdoResultset** can contain columns from one or more tables in a database. Membership of

a keyset **rdoResultset** is fixed.
–   Dynamic-type **rdoResultset**
–the result of a query that can have updatable rows.   A dynamic-type **rdoResultset** is a dynamic set of rows that you can use to add, change, or delete rows from an underlying database table or tables.   A dynamic **rdoResultset** can contain columns from one or more tables in a database.   Membership of a dynamic-type **rdoResultset** is not fixed.

You can use the methods and properties of the **rdoResultset** object to manipulate data and navigate the rows of a result set.   For example, you can:

–   Use the **Type** property to indicate the type of **rdoResultset** created, and the **Updatable** property indicates whether or not you can change the object's rows.
–   Use the **BOF** and **EOF** properties to see if the current row pointer is positioned beyond either end of the **rdoResultset**.
–   Use the **Bookmark** and **PercentPosition** properties and the **Move**, **MoveNext**, **MovePrevious**, **MoveFirst**, and **MoveLast** methods to reposition the current row in all but forward-only
–type **rdoResultset** objects.   Use the **MoveNext**, **MoveLast** or **Move (>0)** to reposition the current row in forward-only type **rdoResultset** objects.
–   Use the **Bookmarkable, Transactions**, and **Restartable** properties to determine if the **rdoResultset** supports bookmarks or transactions, or can be restarted.
–   Use the **LockEdits** property to set the type of locking used to update the **rdoResultset**.
–   Use the **RowCount** property to determine how many rows in the **rdoResultset** are available.
–   Use the **AddNew**, **Edit**, **Update**, and **Delete** methods to add new rows or otherwise modify updatable **rdoResultset** objects.   Use the **CancelUpdate** method to cancel pending edits.
–   Use the **Requery** method to restart the query used to create an **rdoResultset** object.   This method can be used to re-execute a parameterized query.
–   Use the **MoreResults** method to complete processing of the current **rdoResultset** and begin processing the next result set generated from a query.
–   Use the **Cancel** method to terminate processing of an **rdoResultset** object query.

When you create an **rdoResultset**, the current row is positioned to the first row if there are any rows.   If there are no rows, the **RowCount** property setting is 0, and the **BOF** and **EOF** property settings are both **True**.

An **rdoResultset** may not be updatable even if you request an updatable **rdoResultset**. If the underlying database, table, or column isn't updatable, or if your user does not have update permission, all or portions of your **rdoResultset** may be read-only.   Examine the **rdoConnection, rdoResultset,** and **rdoColumn** objects' **Updatable** property to determine if your code can change the rows.

The default collection of an **rdoResultset** is the **rdoColumns** collection, and the default property of an **rdoColumn** object is the **Value** property.   You can simplify your code by taking advantage of these defaults.   For example, the following lines of code all set the value of the PubID column   in the current row of an **rdoResultset**:

```
MyRs.rdoColumns("PubID").Value = 99
MyRs("PubID") = 99
MyRs!PubID = 99
```

You refer to an **rdoResultset** object by its **Name** property setting using this syntax:

rdoResultsets("*name***")**

or

rdoResultsets!*name*

You can also refer to **rdoResultset** objects by their position in the **rdoResultsets** collection using this syntax (where *n* is the *n*th member of the zero-based **rdoResultsets** collection):

rdoResultsets(*n*)

Note To use the Remote Data Objects (RDO), you must set a reference to the Microsoft Remote Data Object 1.0 object library in the Visual Basic References dialog box.

See Also

BOF, **EOF** Properties
Move Method
MoveFirst, **MoveLast**, **MoveNext**, **MovePrevious** Methods
OpenResultset Method
rdoColumn Object, **rdoColumns** Collection
rdoConnection Object, **rdoConnections** Collection
rdoPreparedStatement Object, **rdoPreparedStatements** Collection
rdoTable Object, **rdoTables** Collection
RemoteData Control
Resultset Property
Type Property
Understanding Cursors

rdoResultset Object, rdoResultsets Collection Summary

rdoResultset Object

The **rdoResultset** object contains these collections, methods, and properties.

Collections

rdoColumns (Default)

Methods

AddNew
Cancel
CancelUpdate
Close
Delete
Edit
GetRows
MoreResults
Move
MoveFirst, MoveLast, MoveNext, MovePrevious
Requery
Update

Properties

AbsolutePosition
BOF
Bookmark
Bookmarkable
EOF
hStmt
LastModified
LockEdits
Name
PercentPosition
Restartable
RowCount
StillExecuting
Transactions
Type
Updatable

rdoResultsets Collection

The **rdoResultsets** collection contains one method and one property.

Method

Item

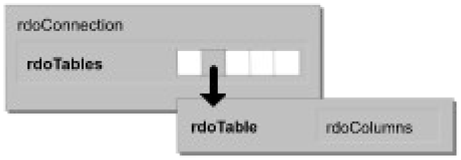
Property

Count

rdoTable Object, rdoTables Collection

See Also Properties Methods Summary

- An **rdoTable** object represents the stored definition of a base table or an SQL view.
- The **rdoTables** collection contains all stored **rdoTable** objects in a database.



Remarks

You map a table definition using an **rdoTable** object and determine the characteristics of an **rdoTable** object using its methods and properties. For example, you can:

- Examine the column properties of any table in an ODBC database. (Note that all **rdoTable** object properties are read-only.)
- Use the **OpenResultset** method to create an **rdoResultset** object based on all of the rows of the base table.
- Use the **Name** property to determine the name of the table or view.
- Use the **RowCount** property to determine the number of rows in the table or view. For base tables, the **RowCount** property contains the number of rows in the specified database table.
- Use the **Type** property to determine the type of table. The ODBC data source driver determines the supported table types.
- Use the **Updatable** property to determine if the table supports changes to its data.

You cannot reference the **rdoTable** objects until you have populated the **rdoTables** collection because it is not automatically populated when you connect to a data source. To populate the **rdoTables** collection, use the **Refresh** method or reference individual members of the collection by their ordinal number.

When you use the **OpenResultset** method against an **rdoTable** object, RDO executes a "SELECT * FROM *table*" query that returns *all* rows of the table using the cursor type specified. By default, a forward-only cursor is created.

You cannot define new tables or change the structure of existing tables using RDO or the **RemoteData** control. To change the structure of a database or perform other administrative functions, use SQL queries or the administrative tools that are provided with the database.

The default collection of an **rdoConnection** object is the **rdoTables** collection, and the default collection of an **rdoTable** object is the **rdoColumns** collection. The default property of an **rdoTable** is the **Name** property. You can simplify your code by using these defaults. For example, the following statements are identical in that they both print the number corresponding to the column data type of a column in an **rdoTable** using a **RemoteData** control:

```
Print RemoteData1.Connection.rdoTables("Publishers").rdoColumns("PubID").Type
Print RemoteData1.Connection("Publishers")("PubID").Type
```

The **Name** property of an **rdoTable** object isn't the same as the name of an object variable to which it's assigned – it is derived from the name of the base table in the database.

You refer to an **rdoTable** object by its **Name** property setting using this syntax:

rdoTables("*name***")**

or

rdoTables!*name*

You can also refer to **rdoTable** objects by their position in the **rdoTables** collection using this syntax (where *n* is the *n*th member of the zero-based **rdoTables** collection):

rdoTables(*n*)

Note To use the Remote Data Objects (RDO), you must set a reference to the Microsoft Remote Data Object 1.0 object library in the Visual Basic References dialog box.

See Also

Name Property
OpenResultset Method
rdoColumn Object, **rdoColumns** Collection
rdoConnection Object, **rdoConnections** Collection
rdoResultset Object, **rdoResultsets** Collection
RowCount Property
Type Property
Updatable Property

rdoTable Object, rdoTables Collection Summary

rdoTable Object

An **rdoTable** contains these collections, methods, and properties.

Collections

rdoColumns (default)

Methods

OpenResultset
Refresh

Properties

Name
RowCount
Type
Updatable

rdoTables Collection

An **rdoTables** collection contains these methods and properties.

Methods

Refresh
Item

Property

Count

Remote Data Objects and Collections

See Also

Remote Data objects and collections provide a framework for using code to create and manipulate components of a remote ODBC database system. Objects and collections have properties that describe the characteristics of database components and methods that you use to manipulate them. Using the containment framework, you create relationships among objects and collections, and these relationships represent the logical structure of your database system.

Objects and collections provide different types of containment relationships: Objects contain zero or more collections, all of different types; and collections contain zero or more objects, all of the same type. Although objects and collections are similar entities, the distinction differentiates the two types of relationships.

Note The RDO is only supported on 32-bit operating systems such as Windows 95 and Windows NT. To use the Remote Data Objects, you must set a reference to the Microsoft Remote Data Object 1.0 object library in the Visual Basic References dialog box.

In the following table, the type of collection in the first column contains the type of object in the second column. The third column describes what each type of object represents.

Collection	**Object**	**Description**
rdoConnections	**rdoConnection**	An open connection
None	**rdoEngine**	The remote database engine
rdoErrors	**rdoError**	Information about any errors associated with this object
rdoEnvironments	**rdoEnvironment**	A logical set of **rdoConnection** objects with a common user name and password
rdoColumns	**rdoColumn**	A column that is part of an **rdoResultset**
rdoParameters	**rdoParameter**	A parameter for an **rdoPreparedStatement**
rdoPreparedStatements	**rdoPreparedStatements**	A saved query definition
rdoResultsets	**rdoResultset**	The rows resulting from a query
rdoTables	**rdoTable**	A table definition

See Also

Remote Data Object Model
Remote Data Objects Event Summary
Remote Data Objects Method Summary
Remote Data Objects Property Summary

RemoteData Control

See Also Properties Methods Events

Provides access to data stored in a remote ODBC data source. The **RemoteData** control enables you to move from row to row in a result set and to display and manipulate data from the rows in bound controls.

Syntax

RemoteData

Remarks

The **RemoteData** control provides an interface between Remote Data Objects (RDO) and data-aware bound controls. With the **RemoteData** control, you can:

- Establish a connection to a data source based on its properties.
- Create an **rdoResultset**.
- Pass the current row's data to corresponding bound controls.
- Permit the user to position the current row pointer.
- Pass any changes made to the bound controls back to the data source.

Overview

Without a **RemoteData** control, a **Data** control or its equivalent, data-aware (bound) controls on a form can't automatically access data. You can perform most remote data access operations using the **RemoteData** control without writing any code at all. Data-aware controls bound to a **RemoteData** control automatically display data from one or more columns for the current row or, in some cases, for a set of rows on either side of the current row. The **RemoteData** control performs all operations on the current row.

If the **RemoteData** control is instructed to move to a different row, all bound controls automatically pass any changes to the **RemoteData** control to be saved to the ODBC data source. The **RemoteData** control then moves to the requested row and passes back data from the current row to the bound controls where it's displayed.

The **RemoteData** control automatically handles a number of contingencies including empty result sets, adding new rows, editing and updating existing rows, and handling some types of errors. However, in more sophisticated applications, you need to trap some error conditions that the **RemoteData** control can't handle. For example, if the remote server has a problem accessing the data source, the user doesn't have permission, or the query can't be executed as coded, a trappable error results. If the error occurs before your application procedures start, or as a result of some internal errors, the Error event is triggered.

Operation

Use the **RemoteData** control properties to describe the data source, establish a connection, and specify the type of cursor to create. If you alter these properties once the result set is created, use the **Refresh** method to rebuild the underlying **rdoResultset** based on the new property settings.

You can manipulate the **RemoteData** control with the mouse – you can move from row to row, or to the beginning or end of the **rdoResultset** by clicking the control. As you manipulate the **RemoteData** control buttons, the current row pointer is repositioned in the **rdoResultset**. You cannot move off either end of the **rdoResultset** using the mouse. You also can't set focus to the **RemoteData** control.

You can also use the objects created by the **RemoteData** control to create additional **rdoConnection,** **rdoResultset**, or **rdoPreparedStatement** objects.

You can set the **RemoteData** control **Resultset** property to an **rdoResultset** created independently of the control. If this is done, the **RemoteData** control properties are reset based on the new **rdoResultset** and **rdoConnection**.

You can set the **Options** property to enable asynchronous creation of the **rdoResultset**.

The Validate event is triggered before each reposition of the current row pointer. You can choose to accept the changes made to bound controls or cancel the operation using the Validate event's *action* argument.

The **RemoteData** control can also manage what happens when you encounter an **rdoResultset** with no rows.   By changing the **EOFAction** property, you can program the **RemoteData** control to enter **AddNew** mode automatically.

Programmatic Operation

To create an **rdoResultset** programmatically with the **RemoteData** control:

– Set the **RemoteData** control properties to describe the desired characteristics of the **rdoResultset**.

– Use the **Refresh** method to begin the automated process or to create the new **rdoResultset**.   Any existing **rdoResultset** is discarded.

All of the **RemoteData** control properties and the new **rdoResultset** object may be manipulated independently of the **RemoteData** control–with or without bound controls.   The **rdoConnection** and **rdoResultset** objects each have properties and methods of their own that can be used with procedures that you write.

For example, the **MoveNext** method of an **rdoResultset** object moves the current row to the next row in the **rdoResultset**.   To invoke this method with an **rdoResultset** created by a **RemoteData** control, you could use this code:

```
RemoteData1.Resultset.MoveNext
```

See Also

Choosing a Cursor Type
Remote Data Objects Overview
Repositioning the Current Row Pointer
Running Asynchronous Queries
Using Bound Controls with the **RemoteData** Control
Using ODBC Handles
Using the **RemoteData** Control

RemoteData Control Events

DragDrop
DragOver
Error
MouseDown
MouseMove
MouseUp
QueryCompleted
Reposition
Validate

RemoteData Control Methods

BeginTrans
Cancel
CommitTrans
Drag
Move
Refresh
RollbackTrans
ShowWhatsThis
UpdateControls
UpdateRow
ZOrder

_

RemoteData Control Properties

AboutBox
Align
Appearance
BackColor
BOFAction
Caption
Connection
Connect
Container
CursorDriver
DataSourceName
DragIcon
DragMode
EditMode
Enabled
Environment
EOFAction
ErrorThreshold
Font
ForeColor
Height
HelpContextID
Index
KeysetSize
Left
LockType
LoginTimeout
LogMessages
MaxRows
MouseIcon
MousePointer
Name
Negotiate
Object
Options
Parent
Password
Prompt
QueryTimeout
ReadOnly
Resultset
ResultsetType
RowsetSize
SQL
StillExecuting

Tag
Top
Transactions
UserName
Version
Visible
WhatsThisHelpID
Width

AbsolutePosition Property (Remote Data)

See Also Applies To

Returns or sets the absolute row number of an **rdoResultset** object's current row.

Syntax

object.**AbsolutePosition** [**=** *value*]

The **AbsolutePosition** 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 from 0 to the number of rows in the **rdoResultset**. Corresponds to the ordinal position of the current row in the **rdoResultset** specified by *object*.

Remarks

Use the **AbsolutePosition** property to position the current row pointer to a specific row based on its ordinal position in a keyset-, dynamic-, or static-type **rdoResultset**. You can also determine the current row number by checking the **AbsolutePosition** property setting.

The **AbsolutePosition** property value is zero-based–a setting of 0 refers to the first row in the **rdoResultset**. Setting a value greater than the number of populated rows causes a trappable error. You can determine the number of populated rows in the **rdoResultset** by checking the **RowCount** property setting.

If there is no current row, as when there are no rows in the **rdoResultset**, -1 is returned. If the current row is deleted, the **AbsolutePosition** property value isn't defined and a trappable error occurs if it is referenced. New rows are added to the end of the sequence if the type of cursor includes dynamic membership.

Note This property isn't intended to be used as a surrogate row number. Using bookmarks is still the recommended way of retaining and returning to a given position. Also, there is no assurance that a given row will have the same absolute position if the **rdoResultset** is re-created again because the order of individual rows within an **rdoResultset** isn't guaranteed unless it's created with an SQL statement using an ORDER BY clause.

The **AbsolutePosition** property isn't available on a forward-only–type **rdoResultset**.

_

AbsolutePosition Property (Remote Data) Applies To

rdoResultset Object

See Also

Bookmark Property
PercentPosition Property
rdoResultset Object, **rdoResultsets** Collection
RowCount Property

AllowZeroLength Property (Remote Data)

See Also Applies To

Returns a value that indicates whether a zero-length string ("") is a valid setting for the **Value** property of an **rdoColumn** object with an **rdTypeCHAR**, **rdTypeVARCHAR**, or **rdTypeLONGVARCHAR** data type.

Syntax

*object***.AllowZeroLength**

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

Return Values

The **AllowZeroLength** property return values are:

Value	Description
True	A zero-length string is a valid value.
False	A zero-length string isn't a valid value.

Remarks

If **AllowZeroLength** is **False** for a column, you must use **Null** to represent "unknown" states – you cannot use empty strings.

See Also

rdoColumn Object, **rdoColumns** Collection
rdoPreparedStatement Object, **rdoPreparedStatements** Collection
rdoResultset Object, **rdoResultsets** Collection
Required Property
Value Property

AllowZeroLength Property (Remote Data) Applies To

rdoColumn Object

AsyncCheckInterval Property (Remote Data)

See Also Applies To

Returns or sets a value specifying the number of milliseconds that RDO waits between checks to see if an asynchronous query is complete.

Syntax

object.**AsyncCheckInterval** [**=** *value*]

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

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.
value	A **Long** expression as described in Remarks.

Remarks

When you use the **rdAsyncEnable** option to execute a query asynchronously, RDO polls the data source periodically to determine if the query has completed. You can change the duration of time between checks by using the **AsyncCheckInterval** property.

The **AsyncCheckInterval** property defaults to 500 milliseconds (twice a second).

Polling too often can adversely affect both server and workstation performance. Polling less frequently can improve performance, but may affect how quickly data is made available to the user.

As long as the asynchronous query is executing, the **StillExecuting** property returns **True**. You can also interrupt and end an asynchronous query by using the **Cancel** method.

See Also

- **Cancel** Method
- **Execute** Method
- **Options** Property
- QueryCompleted Event
- **StillExecuting** Property

AsyncCheckInterval Property (Remote Data) Applies To

rdoConnection Object

Attributes Property (Remote Data)

See Also Applies To

Returns a value that indicates one or more characteristics of an **rdoColumn** object.

Syntax

*object***.Attributes**

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

Return Values

The **Attributes** property return value specifies characteristics of the column represented by the **rdoColumn** object and can be a sum of these constants:

Constant	Value	Description
rdFixedColumn	1	The column size is fixed (default for numeric columns).
rdVariableColumn	2	The column size is variable (text columns only).
rdAutoIncrColumn	16	The column value for new rows is automatically incremented to a unique **Long** integer that can't be changed.
rdUpdatableColumn	32	The column value can be changed.

Remarks

When checking the setting of this property, you can use the **And** operator to test for a specific attribute. For example, to determine whether an **rdoColumn** object is fixed-size, you can use code like the following:

```
If MyResultset![ColumnName].Attributes And rdFixedColumn Then...
```

See Also

rdoColumn Object, **rdoColumns** Collection
rdoTable Object, **rdoTables** Collection

_

Attributes Property (Remote Data) Applies To

rdoColumn Object

BindThreshold Property

See Also Applies To

Returns or sets a value specifying the largest column that will be automatically bound under ODBC.

Syntax

object.**BindThreshold** [**=** *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A **Long** expression as described in Remarks.

Remarks

Several data types support sizes that are far too large to handle using conventional string techniques. For these columns, you must use the **GetChunk** and **AppendChunk** methods. By setting the **BindThreshold** property, you can set the maximum size of chunk that RDO automatically binds to strings. Columns larger than the **BindThreshold** value require use of the **GetChunk** method to retrieve data. The **ChunkRequired** property indicates if the column requires use of **AppendChunk** and **GetChunk** methods.

The default value for **BindThreshold** is 1024 bytes.

See Also

AppendChunk Method
ChunkRequired Property
ColumnSize Method
GetChunk Method

_

BindThreshold Property (Remote Data) Applies To

rdoPreparedStatement Object

BOF, EOF Properties (Remote Data)

See Also Applies To

– **BOF**
–returns a value that indicates whether the current row position is before the first row in an **rdoResultset**.
– **EOF**
–returns a value that indicates whether the current row position is after the last row in an **rdoResultset**.

Syntax

object.**BOF**

object.**EOF**

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

Return Values

The **BOF** property return values are:

Value	Description
True	The current row position is before the first row.
False	The current row position is on or after the first row.

The **EOF** property return values are:

Value	Description
True	The current row position is after the last row.
False	The current row position is on or before the last row.

Remarks

The **BOF** and **EOF** return values are determined by the location of the current row pointer. If either **BOF** or **EOF** is **True**, there is no current row.

You can use the **BOF** and **EOF** properties to determine whether an **rdoResultset** object contains rows or whether you've gone beyond the limits of an **rdoResultset** as you move from row to row.

If you open an **rdoResultset** containing no rows, **BOF** and **EOF** are set to **True**, and the result set's **RowCount** property setting is 0. When you open an **rdoResultset** that contains at least one row, the first row is the current row and **BOF** and **EOF** are **False**; they remain **False** until you move beyond the beginning or end of the **rdoResultset** using the **MovePrevious** or **MoveNext** method, respectively. When you move beyond the beginning or end of the **rdoResultset**, there is no current row.

If you delete the last remaining row in the **rdoResultset** object, **BOF** and **EOF** might remain **False** until you attempt to reposition the current row.

If you use the **MoveLast** method on an **rdoResultset** containing rows, the last row becomes the current row; if you then use the **MoveNext** method, the current row becomes invalid and **EOF** is set to **True**. Conversely, if you use the **MoveFirst** method on an **rdoResultset** containing rows, the first row becomes the current row; if you then use the **MovePrevious** method, there is no current row and **BOF** is set to **True**.

Typically, when you work with all the rows in an **rdoResultset**, your code will loop through the rows using **MoveNext** until the **EOF** property is set to **True**.

If you use **MoveNext** while **EOF** is set to **True** or **MovePrevious** while **BOF** is set to **True**, an error occurs.

This table shows which *Move* methods are allowed with different combinations of **BOF** and **EOF**.

	MoveFirst, MoveLast	**MovePrevious, Move** < 0	**Move** 0	**MoveNext, Move** > 0

BOF=True, EOF=False	Allowed	Error	Error	Allowed
BOF=False, EOF=True	Allowed	Allowed	Error	Error
Both **True**	Error	Error	Error	Error
Both **False**	Allowed	Allowed	Allowed	Allowed

Allowing a *Move* method doesn't mean that the method will successfully locate a row. It merely indicates that an attempt to perform the specified *Move* method is allowed and won't generate an error. The state of the **BOF** and **EOF** properties may change as a result of the attempted Move.

Effect of specific methods on **BOF** and **EOF** settings:

– An **OpenResultset** method internally invokes a **MoveFirst**. Therefore, an **OpenResultset** on an empty set of rows results in **BOF** and **EOF** being set to **True**.

– All *Move* methods that successfully locate a row clear (set to **False**) both **BOF** and **EOF**.

– For dynamic-type **rdoResultset** objects, any **Delete** method, even if it removes the only remaining row from an **rdoResultset**, won't change the setting of **BOF** or **EOF**.

– For other types of **rdoResultset** objects, the **BOF** and **EOF** properties are unchanged as add and delete operations are made because result set membership is fixed.

See Also

BOFAction, **EOFAction** Properties

MoveFirst, **MoveLast**, **MoveNext**, **MovePrevious** Methods

rdoResultset Object, **rdoResultsets** Collection

RemoteData Control

RowCount Property

_

BOF, EOF Properties (Remote Data) Apply To

rdoResultset Object

BOFAction, EOFAction Properties (Remote Data)

See Also Applies To

Returns or sets a value indicating what action the **RemoteData** control takes when the **BOF** or **EOF** property is **True**.

Syntax

object.**BOFAction** [= *value*]

object.**EOFAction** [= *value*]

The **BOFAction** and **EOFAction** property syntaxes have these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A constant or value that specifies an action, as described in Settings.

Settings

For the **BOFAction** property, the settings for *value* are:

Constant	Value	Description
rdMoveFirst	0	**MoveFirst** (Default): Keeps the first row as the current row.
rdBOF	1	**BOF**: Moving past the beginning of an **rdoResultset** triggers the **RemoteData** control's Validate event on the first row, followed by a Reposition event on the invalid (**BOF**) row. At this point, the Move Previous button on the **RemoteData** control is disabled.

For the **EOFAction** property, the settings for *value* are:

Constant	Value	Description
rdMoveLast	0	**MoveLast** (Default): Keeps the last row as the current row.
rdEOF	1	**EOF**: Moving past the end of an **rdoResultset** triggers the **RemoteData** control's Validation event on the last row, followed by a Reposition event on the invalid (**EOF**) row. At this point, the Move Next button on the **RemoteData** control is disabled.
rdAddNew	2	**AddNew**: Moving past the last row triggers the **RemoteData** control's Validation event to occur on the current row, followed by an automatic **AddNew**, followed by a Reposition event on the new row.

Remarks

If you set the **EOFAction** property to **rdAddNew**, once the user moves the current row pointer to **EOF** using the **RemoteData** control, the current row is positioned to a new row in the copy buffer. At this point you can edit the newly added row. If you make changes to the new row and the user subsequently moves the current row pointer using the **RemoteData** control, the row is automatically appended to the **rdoResultset**. If you don't make changes to this new row, and reposition the current row to another row, the new row is discarded. If you use the **RemoteData** control to position to another row while it is positioned over this new row, another new row is created.

When you use code to manipulate **rdoResultset** objects created with the **RemoteData** control, the **EOFAction** property has no effect–it only takes effect when manipulating the **RemoteData** control with the mouse.

In situations where the **RemoteData** control **rdoResultset** is returned with no rows, or after the last row has been deleted, using the **rdAddNew** option for the **EOFAction** property greatly simplifies your code because a new row can always be edited as the current row.

See Also

AddNew Method
BOF, **EOF** Properties
Edit Method
MoveFirst, **MoveLast**, **MoveNext**, **MovePrevious** Methods
rdoResultset Object, **rdoResultsets** Collection
Reposition Event
RowCount Property
SQL Property
Type Property
Validate Event

_

BOFAction, EOFAction Properties (Remote Data) Apply To

RemoteData Control

Bookmark Property (Remote Data)

See Also Applies To

Returns or sets a bookmark that uniquely identifies the current row in an **rdoResultset** object or sets the current row in an **rdoResultset** to a valid bookmark.

Syntax

object.**Bookmark** [**=** *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A **Variant** expression that evaluates to a valid bookmark.

Remarks

When a non-forward-only–type **rdoResultset** object is created or opened, each of its rows already has a unique bookmark. You can save the bookmark for the current row by assigning the value of the **Bookmark** property to a variable. To quickly return to that row at any time after moving to a different row, set the **rdoResultset** object's **Bookmark** property to the value of that variable.

There is no limit to the number of bookmarks you can establish. To create a bookmark for a row other than the current row, move to the desired row and assign the value of the **Bookmark** property to a **Variant** variable that identifies the row.

To make sure the **rdoResultset** supports bookmarks, inspect the value of its **Bookmarkable** property before you use the **Bookmark** property. If **Bookmarkable** is **False**, the **rdoResultset** doesn't support bookmarks, and using the **Bookmark** property results in a trappable error.

The value of the **Bookmark** property isn't guaranteed to be the same as a row number.

Note The **Bookmark** property doesn't apply to forward-only type **rdoResultset** objects.

See Also

AddNew Method

Bookmarkable Property

Edit Method

rdoResultset Object, **rdoResultsets** Collection

_

Bookmark Property (Remote Data) Applies To

rdoResultset Object

Bookmarkable Property (Remote Data)

See Also Applies To

Returns a value that indicates whether an **rdoResultset** object supports bookmarks, which you can set using the **Bookmark** property.

Syntax

object.**Bookmarkable**

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

Return Values

The **Bookmarkable** property return values are:

Value	Description
True	The **rdoResultset** supports bookmarks.
False	The **rdoResultset** doesn't support bookmarks.

Remarks

To make sure an **rdoResultset** supports bookmarks, check the **Bookmarkable** property setting before you attempt to set or check the **Bookmark** property.

See Also

Bookmark Property

rdoResultset Object, **rdoResultsets** Collection

_

Bookmarkable Property (Remote Data) Applies To

rdoResultset Object

ChunkRequired Property (Remote Data)

See Also Applies To

Returns a Boolean value that indicates if data must be accessed using the **GetChunk** method.

Syntax

object.**ChunkRequired**

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

Return Values

The **ChunkRequired** property return values are:

Value	Description
True	The data should be accessed using the **GetChunk** method.
False	The data should not be accessed using the **GetChunk** method.

Remarks

Use the **ChunkRequired** property to determine if the column in question must be manipulated using the **AppendChunk** and **GetChunk** methods. A trappable error is triggered if the **ChunkRequired** property is **True** and you do not use the **AppendChunk** and **GetChunk** methods to manipulate the specified column.

By setting the **BindThreshold** property, you can adjust the number of bytes that force use of the **AppendChunk** and **GetChunk** methods. You can also determine the length of a *chunk* column by using the **ColumnSize** method.

See Also

AppendChunk Method
BindThreshold Property
ColumnSize Method
GetChunk Method
Type Property

ChunkRequired Property (Remote Data) Applies To

rdoColumn Object

Connect Property (Remote Data)

See Also Applies To

Returns or sets a value that provides information about the source of an open **rdoConnection**. When used with the **rdoPreparedStatement** or **rdoConnection** object, this property is read-only. When used with the **RemoteData** control, this property is read-write.

Syntax

*object***.Connect** [**=** *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A string expression as described in Remarks. (Data type is **String**.)

Settings

The **Connect** property return value is a **String** expression composed of zero or more parameters separated by semicolons, as described in Remarks.

Remarks

The **Connect** property is used to pass additional information to and from the ODBC driver manager to establish a connection with a data source. The **Connect** property holds the ODBC connect string which is also used as an argument to the **OpenConnection** method. Once a connection is made, the **Connect** property is completed with the values supplied by the user and the ODBC driver manager. The **Connect** property of the **rdoPreparedStatement** contains this amended connect string.

The following table details valid parameters and typical usage.

Parameter	Specifies	Example
DSN	Registered ODBC data source by name	`DSN=MyDataSource;`
UID	User name of a recognized user of the database	`UID=Victoria;`
PWD	Password associated with user name	`PWD=ChemMajor;`
DRIVER	Description of driver	`DRIVER=SQL Server;`
DATABASE	Default database to use once connected	`DATABASE=Pubs;`

Note Some ODBC drivers require different parameters not shown in this list.

For example, to set the **Connect** property of a **RemoteData** control, you could use code like the following:

```
Dim Cnct As String
Cnct$ = "DSN=WorkData;UID=Chrissy;PWD=MIDFLD;DATABASE=WorkDB;"
RemoteData1.Connect = Cnct$
RemoteData1.SQL = "Select Name, City From Teams Where Type = 12"
RemoteData1.Refresh
```

You can use this same connect string to establish a new connection:

```
Dim Cn As rdoConnection
Set Cn = rdoEnvironments(0).OpenConnection("",rdDriverNoPrompt,True,Cnct$)
```

Note Valid parameters are determined by the ODBC driver. The parameters shown in the preceding example are supported by the Microsoft SQL Server ODBC driver.

LOGINTIMEOUT and DBQ are not valid parameters of the **RemoteData** control or the **rdoConnection** object's **Connect** property.   These parameters are supported by the Microsoft Jet database engine, and not by the ODBC driver.   To set login timeout delay, you must use the **LoginTimeout** property of the **rdoEngine** object.

If the connect string is null, the information provided by the DSN is incomplete, or invalid arguments provided for the connection cannot be established.   If your code sets the *prompt* argument of the **OpenConnection** method or the **RemoteData** control's **Prompt** property to prohibit user completion of missing ODBC connect arguments, a trappable error is triggered.   Otherwise, a dialog box listing all registered ODBC data source names is displayed by the ODBC driver so the user can select a data source.   Once the user chooses a valid DSN from the list presented, other missing information is collected to complete the connection including user name and password.

If a password is required, but not provided in the **Connect** property setting, a login dialog box is displayed the first time a table is accessed by the ODBC driver and each time the connection is closed and reopened.

When connecting to ODBC data sources that support domain-managed security, set the UID and PWD parameters to "".   In this case, the Windows NT user name and password are passed to the data source for validation.   This strategy permits access to the data source by users with access to the NT domain through authenticated workstation logons.

You can set the **Connect** property for an **rdoConnection** object by providing a *connect* argument to the **OpenConnection** method.   Once the connection is established, you can check the **Connect** property setting to determine the DSN, database, user name, password, or ODBC data source of the database.

See Also

Connection Property
LoginTimeout Property
OpenConnection Method
rdoConnection Object, **rdoConnections** Collection
rdoEngine Object
rdoPreparedStatement Object, rdoPreparedStatements Collection
rdoTable Object, **rdoTables** Collection

Connect Property (Remote Data) Applies To

rdoConnection Object
rdoPreparedStatement Object
RemoteData Control

Connection Property (Remote Data)

See Also Applies To

Returns a reference to a **RemoteData** control's underlying **rdoConnection** object.

Syntax

*object***.Connection**

Set *connection* **=** *object***.Connection**

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

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

Remarks

When a **RemoteData** control is initialized, RDO opens a connection to the data source specified in the control's **Connect** property. The **rdoConnection** object created by RDO is exposed by the **Connection** property.

rdoConnection objects have properties and methods you can use to manage data. You can use any method of an **rdoConnection** object, such as **Close** and **Execute**, with the **Connection** property of a **RemoteData** control. You can also examine the internal structure of the database by using its **rdoTables** collection, and in turn, the columns of individual **rdoTable** objects.

_
Connection Property (Remote Data) Applies To

RemoteData Control

See Also

Close Method
Connect Property
DataSourceName Property
Execute Method
Name Property
rdoColumn Object, **rdoColumns** Collection
rdoConnection Object, **rdoConnections** Collection
rdoResultset Object, **rdoResultsets** Collection
rdoTable Object, **rdoTables** Collection
RemoteData Control

Count Property (Remote Data)

See Also Applies To

Returns the number of members in an RDO collection.

Syntax

object.**Count**

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

Return Values

The **Count** property setting is a **Long** integer.

Remarks

Use the **Count** property to determine the number of members in a collection. For example, to see how many tables are in a database, examine the **Count** property of the **rdoTables** collection.

See Also

rdoColumn Object, **rdoColumns** Collection
rdoConnection Object, **rdoConnections** Collection
rdoEnvironment Object, **rdoEnvironments** Collection
rdoError Object, **rdoErrors** Collection
rdoParameter Object, **rdoParameters** Collection
rdoPreparedStatement Object, **rdoPreparedStatements** Collection
rdoResultset Object, **rdoResultsets** Collection
rdoTable Object, **rdoTables** Collection

Count Property (Remote Data) Applies To

rdoColumns Collection
rdoConnections Collection
rdoEnvironments Collection
rdoErrors Collection
rdoParameters Collection
rdoPreparedStatements Collection
rdoResultsets Collection
rdoTables Collection

CursorDriver Property (Remote Data)

See Also Applies To

Returns or sets a value that specifies the type of cursor to be created.

Syntax

object.**CursorDriver** [**=** *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	An **Integer** or constant as described in Settings.

Settings

Constant	Value	Description
rdUseIfNeeded	0	The ODBC driver will choose to use the appropriate style of cursors. It will use server-side cursors if they are available.
rdUseOdbc	1	The RDO layer will use the ODBC cursor library. This gives better performance for small result sets but degrades quickly for larger result sets.
rdUseServer	2	Use server-side cursors. For most large operations this will give better performance, but can cause more network traffic.

Remarks

These constants are also used to set the default cursor driver in the **rdoEngine**. This property only affects connections established after the **CursorDriver** property has been set. Changing the **CursorDriver** property has no effect on existing connections.

See Also

rdoConnection Object, **rdoConnections** Collection
rdoDefaultCursorDriver Property
rdoEngine Object
rdoEnvironment Object, **rdoEnvironments** Collection

CursorDriver Property (Remote Data) Applies To

rdoEnvironment Object
RemoteData Control

DataSourceName Property (Remote Data)

See Also Applies To

Returns or sets the data source name for a **RemoteData** control.

Syntax

object.**DataSourceName** [**=** *datasourcename*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
datasourcename	A string expression that indicates the registered data source name.

Remarks

This property can be left blank if the control's **Connect** property identifies a data source name (DSN) registered in the ODBC.INI file (16-bit) or Windows Registry (32-bit).

Once the **rdoConnection** is opened by the **RemoteData** control, the **DataSourceName** property contains the DSN used to establish the connection − it may be different from the value set before the connection is opened, because a user might select a data source from a list of valid DSN entries during the connection process.

If you change this property after the control's **rdoConnection** object is open, you must use the **Refresh** method to open a new connection to the data source.

_

DataSourceName Property (Remote Data) Applies To

RemoteData Control

See Also

Connect Property
OpenConnection Method
rdoEngine Object
rdoRegisterDataSource Method
Refresh Method
RemoteData Control

Description Property (Remote Data)

See Also Applies To

Returns a descriptive string associated with an error.

Syntax

object.**Description**

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

Return Values

The **Description** property return value is a string expression containing a description of the error.

Remarks

The **Description** property comprises a short description of the error which can be used to alert the user to an error that you cannot or do not want to handle. The **Description** property contains context information about where the error occurred. The SQLState code is prepended to the message, followed by a colon and a space. For example "S0021: Cannot find XXX".

See Also

HelpContext, **HelpFile** Properties
Number Property
rdoError Object, **rdoErrors** Collection
Source Property
SQLRetCode Property
SQLState Property

Description Property (Remote Data) Applies To

rdoError Object

Direction Property (Remote Data)

See Also Applies To

Returns or sets a value indicating how a parameter is passed to or from a procedure.

Syntax

object.**Direction** [= *value*]

The **Direction** 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** as described in Settings.

Settings

The settings for *value* are one of the following values:

Constant	Value	Description
rdParamInput	0	(Default) The parameter is used to pass information to the procedure.
rdParamInputOutput	1	The parameter is used to pass information both to and from the procedure.
rdParamOutput	2	The parameter is used to return information from the procedure as in an output parameter in SQL.
rdParamReturnValue	3	The parameter is used to pass the return value from a procedure.

Remarks

Use the **Direction** property to determine whether the parameter is an input parameter, output parameter, or both–or if the parameter is the return value from the procedure. Some ODBC drivers do not provide information on the direction of parameters to a SELECT statement or procedure call. In these cases, it is necessary to set the direction prior to executing the query.

For example, the following procedure returns a value from a stored procedure:

```
{? = call sp_test}
```

This call produces one parameter – the return value. It is necessary to set the direction of this parameter to **rdParamOutput** or **rdParamReturnValue** before executing the prepared statement. For example:

```
Dim my_statement As rdoPreparedStatement
Set my_statement = someRdoConnection.CreatePreparedStatement _
   ("MyPs", "{?=call sp_testprocedure }", ...)
my_statement.rdoParameters(0).Direction = rdParamOutput
my_statement.Execute
Print my_statement.rdoParameters(0).Value
```

You need to set all parameter directions except **rdParamInput** before accessing or setting the values of the parameters and before executing the **rdoPreparedStatement**.

All parameter directions default to **rdParamInput** unless the ODBC driver supports returning the correct information.

You should use **rdParamReturnValue** for return values, but you can use **rdParamOutput** where **rdParamReturnValue** is not supported.

Note The Microsoft SQL Server 6.0 driver automatically sets the **Direction** property for all procedure parameters.

See Also

Execute Method

rdoParameter Object, **rdoParameters** Collection

rdoPreparedStatement Object, **rdoPreparedStatements** Collection

Direction Property (Remote Data) Applies To

rdoParameter Object

EditMode Property (Remote Data)

See Also Applies To

Returns a value that indicates the state of editing for the current row.

Syntax

object.**EditMode**

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

Return Values

The **EditMode** property returns an **Integer** or constant as described in the following table:

Constant	Value	Description
rdEditNone	0	No editing operation is in progress.
rdEditInProgress	1	The **Edit** method has been invoked, and the current row is in the copy buffer.
rdEditAdd	2	The **AddNew** method has been invoked, and the current row in the copy buffer is a new row that hasn't been saved in the database.

Remarks

The **EditMode** property is most useful when you want to depart from the default functionality of a **RemoteData** control.   You can check the value of the **EditMode** property and the value of the *action* parameter in the Validate event procedure to determine whether to invoke the **Update** method.

You can also check to see if the **LockEdits** property of the **rdoResultset** is **True** and the **EditMode** property setting is **rdEditInProgress** to determine whether the current data page is locked.

See Also

AddNew Method
CancelUpdate Method
Edit Method
LockEdits Property
rdoResultset Object, **rdoResultsets** Collection
Update Method
UpdateRow Method
Validate Event

EditMode Property (Remote Data) Applies To

RemoteData Control

Environment Property (Remote Data)

See Also Applies To

Returns a reference to a **RemoteData** control's underlying **rdoEnvironment** object.

Syntax

object.**Environment**

Set *environment* = *object*.**Environment**

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

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

Remarks

When a **RemoteData** control is initialized, RDO uses the default **rdoEnvironments**(0) – the **Environment** property is initially set to this object.

If you assign another **rdoResultset** to the **RemoteData** control's **Resultset** property, the **Environment** property is set to the **rdoEnvironment** object used to create the result set. **rdoEnvironment** objects have properties and methods you can use to manage data. For example, you can use any method of an **rdoEnvironment** object, such as **OpenConnection**, **BeginTrans**, **CommitTrans**, or **RollbackTrans**, with the **Environment** property.

See Also

BeginTrans, **CommitTrans**, **RollbackTrans** Methods
Connect Property
OpenConnection Method
rdoConnection Object, **rdoConnections** Collection
rdoEnvironment Object, **rdoEnvironments** Collection
rdoResultset Object, **rdoResultsets** Collection
RemoteData Control

Environment Property (Remote Data) Applies To

RemoteData Control

ErrorThreshold Property (Remote Data)

See Also Applies To

Returns or sets a value that determines the severity level that constitutes a fatal error.

Syntax

object.**ErrorThreshold** [**=** *value*]

The **ErrorThreshold** 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 value that specifies a level of error severity as described in Settings. (Data type is **Long**.)

Settings

The setting for *value* is the severity level that RDO uses to determine whether a trappable error should be generated. The default value for **ErrorThreshold** is determined by the **rdoDefaultErrorThreshold** property, and if no value is specified in the **rdoDefaultErrorThreshold** property, the **ErrorThreshold** property defaults to -1, which indicates the **ErrorThreshold** is ignored. The maximum and recommended values are determined by the ODBC driver and data source.

Remarks

Use the **ErrorThreshold** to set the severity level which forces error handlers to consider the error "fatal." All errors and warnings generated by any operation are logged in the **rdoErrors** collection. However, only errors below the error threshold cause the operation to terminate and Visual Basic to issue a trappable error.

See Also

QueryTimeout Property

rdoDefaultErrorThreshold Property

ErrorThreshold Property (Remote Data) Applies To

rdoPreparedStatement Object
RemoteData Control

HelpContext, HelpFile Properties (Remote Data)

See Also Applies To

– **HelpContext**
–returns a context ID for a topic in a Microsoft Windows Help file.
– **HelpFile**
–returns a fully qualified path to the Help file as a variable.

Syntax

object.**HelpContext**

object.**HelpFile**

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

Return Values

The **HelpContext** property returns a **Long** value.

The **HelpFile** property returns a **String** value.

Remarks

If a Microsoft Windows Help file is specified in **HelpFile**, the **HelpContext** property is used to automatically display the Help topic it identifies.

Note You should write routines in your application to handle typical errors. When programming with an object, you can use the Help supplied by the object's Help file to improve the quality of your error handling, or to display a meaningful message to your user if the error is not recoverable.

See Also

Description Property
Number Property
rdoError Object, **rdoErrors** Collection
ShowWhatsThis Method
Source Property
SQLRetCode Property
SQLState Property
WhatsThisHelpID Property

HelpContext, HelpFile Properties (Remote Data) Apply To

rdoError Object
RemoteData Control

hDbc Property (Remote Data)

See Also Applies To

Returns a value corresponding to the ODBC connection handle.

Syntax

object.**hDbc**

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

Return Values

The **hDbc** property returns a **Long** value containing the ODBC connection handle created by the ODBC driver manager corresponding to the specified **rdoConnection** object.

Remarks

This handle can be used to execute ODBC functions that require an ODBC **hDbc** connection handle.

Note While it is possible to execute ODBC API functions using the ODBC **hEnv, hDbc**, and **hStmt** handles, it is recommended that you do so with caution. Improper use of arbitrary ODBC API functions using these handles can result in unpredictable behavior. You should not attempt to save this handle in a variable for use at a later time as the value is subject to change.

See Also

hEnv Property
hStmt Property

hDbc Property (Remote Data) Applies To

rdoConnection Object

hEnv Property (Remote Data)

See Also Applies To

Returns a value corresponding to the ODBC environment handle.

Syntax

object.**hEnv**

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

Return Values

The **hEnv** property returns a **Long** value containing the ODBC environment handle created by the ODBC driver manager corresponding to the specified **rdoEnvironment** object.

Remarks

This handle can be used to execute ODBC functions that require an ODBC **hEnv** environment handle.

Note While it is possible to execute ODBC API functions using the ODBC **hEnv, hDbc**, and **hStmt** handles, it is recommended that you do so with caution. Improper use of arbitrary ODBC API functions using these handles can result in unpredictable behavior. You should not attempt to save this handle in a variable for use at a later time as the value is subject to change.

See Also

hDbc Property

hStmt Property

hEnv Property (Remote Data) Applies To

rdoEnvironment Object

hStmt Property (Remote Data)

See Also Applies To

Returns a value corresponding to the ODBC statement handle.

Syntax

object.**hStmt**

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

Return Values

The **hStmt** property returns a **Long** value containing the ODBC statement handle created by the ODBC driver manager corresponding to the specified **rdoResultset** object.

Remarks

This handle can be used to execute ODBC functions that require an ODBC **hStmt** statement handle.

Note While it is possible to execute ODBC API functions using the ODBC **hEnv, hDbc**, and **hStmt** handles, it is recommended that you do so with caution. Improper use of arbitrary ODBC API functions using these handles can result in unpredictable behavior. You should not attempt to save this handle in a variable for use at a later time as the value is subject to change.

See Also

hDbc Property

hEnv Property

hStmt Property (Remote Data) Applies To

rdoPreparedStatement Object
rdoResultset Object

KeysetSize Property (Remote Data)

See Also Applies To

Returns or sets a value indicating the number of rows in the keyset buffer.

Syntax

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

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

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

Settings

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

Remarks

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

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

Not all ODBC data sources support keyset cursors.

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

See Also

MaxRows Property

RowsetSize Property

_

KeysetSize Property (Remote Data) Applies To

rdoPreparedStatement Object
RemoteData Control

LastModified Property (Remote Data)

See Also   Applies To

Returns a bookmark indicating the most recently added or changed row.

Syntax

object.**LastModified**

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

Return Values

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

Remarks

You can use **LastModified** to move to the most recently added or updated row.

See Also

Bookmark Property

Bookmarkable Property

rdoResultset Object, **rdoResultsets** Collection

LastModified Property (Remote Data) Applies To

rdoResultset Object

LockType Property (Remote Data)

See Also Applies To

Returns or sets a **Long** integer value indicating the type of concurrency handling.

Syntax

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

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

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

Settings

The settings for *value* are:

Constant	Value	Description
rdConcurReadOnly	1	(Default) Cursor is read-only. No updates are allowed.
rdConcurLock	2	Pessimistic concurrency. Cursor uses the lowest level of locking sufficient to ensure the row can be updated.
rdConcurRowVer	3	Optimistic concurrency based on row ID. Cursor compares row ID in old and new rows to determine if changes have been made since the row was last accessed.
rdConcurValues	4	Optimistic concurrency based on row values. Cursor compares data values in old and new rows to determine if changes have been made since the row was last accessed.

Remarks

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

If the concurrency option is not supported by the data source, the driver substitutes a different concurrency option and returns a trappable error (SQLState Code 01S02 "`Option Value Changed`"). For **rdConcurValues**, the driver substitutes **rdConcurRowVer** and vice versa. For **rdConcurLock**, the driver substitutes, in order: **rdConcurRowver** or **rdConcurValues**.

See Also

CreatePreparedStatement Method
OpenResultset Method
rdoPreparedStatement Object, **rdoPreparedStatements** Collection

LockType Property (Remote Data) Applies To

rdoPreparedStatement Object

LockEdits Property (Remote Data)

See Also Applies To

Returns a **Boolean** value indicating the locking that is in effect during editing.

Syntax

object.**LockEdits**

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

Return Values

The return values for **LockEdits** are:

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

Remarks

If a page is locked and the data source uses page locking, no other user can edit rows on the same page. If row-level locking is used, the row being edited is locked. If **LockEdits** is **True** and another user already has the page locked, an error occurs when you use the **Edit** method. Other users can read data from locked pages.

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

Note Data page size is determined by the data source. Microsoft SQL Server uses 2K data pages.

See Also

Bookmark Property
CancelUpdate Method
Close Method
Edit Method
rdoResultset Object, **rdoResultsets** Collection
Update Method

LockEdits Property (Remote Data) Applies To

rdoResultset Object

LoginTimeout Property (Remote Data)

See Also Applies To

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

Syntax

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

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

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

Remarks

If *value* is 0, no timeout occurs.

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

When used with an **rdoEnvironment** object, the **LoginTimeout** property specifies a global value for all login operations associated with the **rdoEnvironment**.

The default timeout value is either 15 seconds or a value set by the **rdoDefaultLoginTimeout** property. The setting of **LoginTimeout** on an **rdoConnection** object overrides the default value.

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

See Also

QueryTimeout Property
rdoDefaultLoginTimeout Property
rdoEngine Object

LoginTimeout Property (Remote Data) Applies To

rdoEnvironment Object
RemoteData Control

LogMessages Property (Remote Data)

See Also Applies To

Returns or sets a value indicating the path of the ODBC trace file created by the ODBC driver manager to record all ODBC operations.

Syntax

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

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

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

Settings

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

Remarks

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

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

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

See Also

rdoResultset Object, **rdoResultsets** Collection

LogMessages Property (Remote Data) Applies To

rdoPreparedStatement Object
RemoteData Control

MaxRows Property (Remote Data)

See Also Applies To

Returns or sets a value indicating the maximum number of rows to be returned from a query.

Syntax

object.**MaxRows** [**=** *value*]

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

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

Settings

The setting for *value* ranges from -1 to any number. If *value* is set to -1, no limit is placed on the number of rows returned (default).

Remarks

This property determines the maximum number of rows returned from a query. Once the number of rows specified by **MaxRows** is returned to your application in an **rdoResultset**, the query processor stops returning additional rows – even if more rows would qualify for inclusion in the result set. This property is useful in situations where limited resources prohibit management of large numbers of result set rows.

See Also

RowsAffected Property

MaxRows Property (Remote Data) Applies To

rdoPreparedStatement Object
RemoteData Control

Name Property (Remote Data)

See Also Applies To

Returns the name of a remote data object.

Syntax

object.**Name**

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

Return Values

The **Name** property returns a string expression that represents the name assigned to the object. The following table describes how each object is assigned its name.

Remote Data Object	Name property is determined by the:
rdoEnvironments(0)	**rdoEngine** –Set to "Default_Environment".
rdoEnvironments(1-*n*)	*name* argument of **rdoCreateEnvironment.**
rdoConnections(0-*n)*	Data source name (DSN) used for connection.
rdoResultsets(0-*n)*	First 256 characters of the SQL query.
rdoPreparedStatements(0-*n*)	*name* argument in **CreatePreparedStatement** method.
rdoTables(0-*n)*	Database table name once **rdoTables** collection is populated.
rdoParameters(0-*n)*	"Param*n*" where "*n*" is the ordinal number.
rdoColumns(0-*n)*	Database column name.
rdoErrors(0-*n*)	Not applicable. **rdoErrors** collection members can only be referenced by their ordinal number.

Remarks

rdoTable and **rdoPreparedStatement** objects can't share the same name.

Use the **Name** property to reference members of a collection in code, but in most cases, it is easier to simply use the ordinal number. Generally, you can use the **Name** property to map database table and column names.

See Also

CreatePreparedStatement Method

Name Property (Remote Data) Applies To

rdoColumn Object
rdoConnection Object
rdoEnvironment Object
rdoError Object
rdoParameter Object
rdoPreparedStatement Object
rdoResultset Object
rdoTable Object
RemoteData Control

Negotiate Property (Remote Data)

See Also Applies To

Sets a value that determines whether a control that can be aligned is displayed when an active object on the form displays one or more toolbars. Not available at run time.

Syntax

object.**Negotiate** [**=** *value*]

The **Negotiate** 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 as described in Settings.

Settings

The **Negotiate** property has these settings:

Setting	Description
True	If the control is aligned within the form (the **Align** property is set to a nonzero value), the control remains visible when an active object on the form displays a toolbar.
False	(Default) The control isn't displayed when an active object on the form displays a toolbar. The toolbar of the active object is displayed in place of the control.

Remarks

The **Negotiate** property exists for all controls with an **Align** property. You use the **Align** property to align the control within a **Form** or **MDIForm** object; however, the toolbar negotiation occurs only on the **MDIForm**. The aligned control must be on the **MDIForm**.

If the **NegotiateToolbars** property is set to **False**, the setting of the **Negotiate** property has no effect.

See Also

Align Property

Negotiate Property (Remote Data) Applies To

RemoteData Control

Number Property (Remote Data)

See Also Applies To

Returns a numeric value specifying a native error.

Syntax

object.**Number**

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

Return Values

The return value is a **Long** integer representing an error number.

Remarks

Number is the **Error** object's default property.

The **Number** property returns the SQL native error number for the **Error** object.

Use the **Number** property to determine the error that occurred. The value of the property corresponds to a unique number that corresponds to an error condition.

Note The SQL Server error severity level is not returned by the ODBC driver, and is therefore unavailable.

See Also

Description Property
HelpContext, **HelpFile** Properties
rdoError Object, **rdoErrors** Collection
Source Property
SQLRetCode Property
SQLState Property

Number Property (Remote Data) Applies To

rdoError Object

Options Property (Remote Data)

See Also Applies To

Returns or sets a value that specifies one or more characteristics of the **rdoResultset** object exposed by the control's **ResultsetType** property.

Syntax

object.**Options** [**=** *value*]

The **Options** 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** as described in Settings.

Settings

Use the following value to set the **Options** property for the **RemoteData** control:

Constant	Value	Description
rdAsyncEnable	32	Execute the query asynchronously.

Remarks

Asynchronous operations permit RDO to work in the background on operations like creating result sets or executing procedures while your foreground code continues to work.

Whenever you use the **OpenResultset**, **Execute**, or **MoreResults** methods with the **rdAsyncEnable** option, control returns immediately to your application − before the operation is completed by RDO. If required, RDO periodically checks the data source to see if the operation is complete. You can adjust the frequency of this polling by setting the **AsyncCheckInterval** property. To see if your operation has completed, check the **StillExecuting** property which remains **True** until RDO completes the operation. To cancel the operation, use the **Cancel** method.

If you change the **Options** property at run time, you must use the **Refresh** method for the change to have any effect.

This property corresponds to the *options* argument in the **OpenResultset** and **Execute** methods.

_
Options Property (Remote Data) Applies To

RemoteData Control

See Also

CreatePreparedStatement Method
Execute Method
MoreResults Method
OpenResultset Method
QueryCompleted Event
ResultsetType Property

OrdinalPosition Property (Remote Data)

See Also Applies To

Returns the relative position of an **rdoColumn** object within the **rdoColumns** collection.

Syntax

*object***.OrdinalPosition**

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

Return Values

The **OrdinalPosition** property return value is an **Integer** expression as described in Remarks.

Remarks

This property indicates the ordinal position of the column within the **rdoColumns** collection.

See Also

rdoColumn Object, **rdoColumns** Collection
rdoPreparedStatement Object, **rdoPreparedStatements** Collection
rdoResultset Object, **rdoResultsets** Collection
rdoTable Object, **rdoTables** Collection
Refresh Method

_

OrdinalPosition Property (Remote Data) Applies To

rdoColumn Object

Password Property (Remote Data)

See Also Applies To

Represents the password used during creation of an **rdoEnvironment** object.

Syntax

object.**Password**

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

Remarks

This property setting is write-only – it may only be specified in the process of creating a new **rdoEnvironment** using the **rdoCreateEnvironment** method.

The password is set when the **rdoEnvironment** is either created automatically by the **RemoteData** control, by the first reference to a remote data object, or when the **rdoCreateEnvironment** method is executed.

The **rdoDefaultPassword** property of the **rdoEngine** object is used as a default if no password is provided. The initial default password is "".

See Also

rdoCreateEnvironment Method
rdoDefaultUser, **rdoDefaultPassword** Properties
rdoEngine Object
rdoEnvironment Object, **rdoEnvironments** Collection
UserName Property

Password Property (Remote Data) Applies To

rdoEnvironment Object
RemoteData Control

PercentPosition Property (Remote Data)

See Also Applies To

Returns or sets a value that indicates or changes the approximate location of the current row in the **rdoResultset** object based on a percentage of the rows in the **rdoResultset**.

Syntax

*object***.PercentPosition** [**=** *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A number between 0.0 and 100.00. (Data type is **Single**.)

Remarks

To indicate or change the approximate position of the current row in an **rdoResultset**, you can check or set the **PercentPosition** property. Before you set or check the **PercentPosition** property, populate the **rdoResultset** by moving to the last row. If you use the **PercentPosition** property before fully populating the **rdoResultset**, the amount of movement is relative to the number of rows accessed – as indicated by the **RowCount** property. You can move to the last row using the **MoveLast** method.

Note Using the **PercentPosition** property to move the current row to a specific row in an **rdoResultset** isn't recommended–the **Bookmark** property or **AbsolutePosition** property is better suited for this task.

Once you set the **PercentPosition** property to a value, the row at the approximate position corresponding to that value becomes current, and the **PercentPosition** property is reset to a value that reflects the approximate position of the current row. For example, if your **rdoResultset** contains only five rows, and you set its **PercentPosition** value to 77, the value returned from the **PercentPosition** property might be 80, not 77.

The **PercentPosition** property applies to keyset-type and dynamic-type **rdoResultset** objects.

You can use the **PercentPosition** property with a scroll bar on a **Form** or **TextBox** to indicate the location of the current row in an **rdoResultset**.

The **PercentPosition** property is not supported by all cursor types and driver combinations. If the setting is not supported, the **PercentPosition** property returns 50. If the position cannot be set, no movement occurs.

See Also

AbsolutePosition Property
Bookmark Property
MoveFirst, **MoveLast**, **MoveNext**, **MovePrevious** Methods
rdoResultset Object, **rdoResultsets** Collection

_

PercentPosition Property (Remote Data) Applies To

rdoResultset Object

Prompt Property (Remote Data)

See Also Applies To

Returns or sets a value that specifies if the ODBC driver manager should prompt for missing connect string arguments.

Syntax

*object***.Prompt** [**=** *value*]

The **Prompt** 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** as described in Settings.

Settings

The settings for the **Prompt** property are:

Constant	Value	Description
rdDriverPrompt	0	The driver manager displays the ODBC Data Sources dialog box. The connection string used to establish the connection is constructed from the data source name (DSN) selected and completed by the user via the dialog boxes. Or, if no DSN is chosen and the **DataSourceName** property is empty, the default DSN is used.
rdDriverNoPrompt	1	The driver manager uses the connection string provided in *connect*. If sufficient information is not provided, the **OpenConnection** method returns a trappable error.
rdDriverComplete	2	If the connection string provided includes the DSN keyword, the driver manager uses the string as provided in *connect*, otherwise it behaves as it does when **rdDriverPrompt** is specified.
rdDriverCompleteRequired	3	(Default) Behaves like **rdDriverComplete** except the driver disables the controls for any information not required to complete the connection.

Remarks

When RDO opens a connection based on the parameters of the **RemoteData** control, the **Connect** property is expected to contain sufficient information to establish the connection. If information like the data source name, user name, or password are not provided, the ODBC driver manager exposes one or more dialog boxes to gather this information from the user. If you do not want these dialog boxes to appear, set the **Prompt** property accordingly to disable this feature.

See Also

OpenConnection Method

Prompt Property (Remote Data) Applies To

RemoteData Control

QueryTimeout Property (Remote Data)

See Also Applies To

Returns or sets a value that specifies the number of seconds the ODBC driver manager waits before a timeout error occurs when a query is executed.

Syntax

object.**QueryTimeout** [**=** *value*]

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

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

Remarks

If *value* is 0, no timeout occurs (default).

When you're accessing an ODBC data source, there may be delays due to network traffic or heavy use of the remote server. Rather than waiting indefinitely, you can specify how long to wait before the ODBC driver manager produces a trappable error. The **QueryTimeout** property is used when you create **rdoResultset** objects or use the **Execute** method.

When used with an **rdoConnection** object, the **QueryTimeout** property specifies a global value for all queries associated with the data source.

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

When you use an **rdoPreparedStatement**, the **rdoConnection** object's **QueryTimeout** property is used as a default value unless you specify a new value in the **rdoPreparedStatement** object's **QueryTimeout** property.

See Also

Execute Method

OpenResultset Method

rdoConnection Object, **rdoConnections** Collection

rdoPreparedStatement Object

_

QueryTimeout Property (Remote Data) Applies To

rdoConnection Object
rdoPreparedStatement Object
RemoteData Control

rdoDefaultCursorDriver Property (Remote Data)

See Also Applies To

Returns or sets the type of ODBC or server cursor used by the ODBC driver manager.

Syntax

object.**rdoDefaultCursorDriver** [= *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	An **Integer** constant or value that specifies a type of ODBC cursor as described in Settings.

Settings

The settings for *value* are:

Constant	Value	Description
rdUseIfNeeded	0	(Default)The ODBC driver will use the appropriate style of cursors. Server-side cursors are used if they are available.
rdUseODBC	1	The RDO layer uses the ODBC cursor library. This option gives better performance for small result sets, but degrades quickly for larger result sets.
rdUseServer	2	The ODBC driver will use server-side cursors. For most large operations this gives better performance, but might cause more network traffic.

Remarks

When server-side cursors are used, the database engine uses its own resources to store keyset values. Data values are still transmitted over the network as with client-side cursors, but the impact on local workstation memory and disk space is reduced.

For SQL Server, server-side cursors are not used if the cursor is read-only and forward-only.

See Also

CursorDriver Property

Type Property

rdoDefaultCursorDriver Property (Remote Data) Applies To

rdoEngine Object

rdoDefaultUser, rdoDefaultPassword Properties (Remote Data)

See Also Applies To

– **rdoDefaultUser**
–returns or sets the default user name assigned to any new **rdoEnvironment**.
– **rdoDefaultPassword**
–returns or sets the default password assigned to any new **rdoEnvironment**.

Syntax

object.**rdoDefaultUser** [**=** *value*]

object.**rdoDefaultPassword** [**=** *value*]

The **rdoDefaultUser** and **rdoDefaultPassword** property syntaxes have these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A string expression that specifies either a user name or password.

Remarks

Unless other values are supplied in the **rdoCreateEnvironment** method, the **rdoDefaultUser** and **rdoDefaultPassword** properties determine the user name and password used when the **rdoEnvironment** object is created. These properties can also return the name used when an **rdoEnvironment** is created.

By default, the *value* for **rdoDefaultUser** and **rdoDefaultPassword** is "" (a zero-length string).

See Also

Password Property
rdoCreateEnvironment Method
UserName Property

rdoDefaultUser, rdoDefaultPassword Properties (Remote Data) Apply To

rdoEngine Object

rdoDefaultErrorThreshold Property (Remote Data)

See Also Applies To

Returns or sets a value that indicates the default value for the **ErrorThreshold** property for **rdoPreparedStatement** objects.

Syntax

object.**rdoDefaultErrorThreshold** [**=** *value*]

The **rdoDefaultErrorThreshold** 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 value that specifies a level of error severity as described in Settings.

Settings

The **rdoDefaultErrorThreshold** property is a **Long** expression. The settings for *value* are:

Setting	Description
Number property > **ErrorThreshold**	Run-time error is not generated.
Number property <= **ErrorThreshold**	Run-time error is generated.
ErrorThreshold = -1	No error threshold is enforced.

Remarks

Use the **rdoDefaultErrorThreshold** property to provide a default value for the **ErrorThreshold** property. If you do not specify a value, **rdoDefaultErrorThreshold** is -1, which indicates no threshold is enforced.

If an error occurs on the data source, it is passed back to the client. If the **Number** property of the error is less than the **ErrorThreshold** property, a trappable error results – otherwise, the function causing the error returns with an **rdSQLSuccessWithInfo** return code.

See Also

ErrorThreshold Property
Number Property
rdoError Object, **rdoErrors** Collection
SQLRetCode Property

rdoDefaultErrorThreshold Property (Remote Data) Applies To

rdoEngine Object

rdoDefaultLoginTimeout Property (Remote Data)

See Also Applies To

Returns or sets a value that determines the number of seconds the ODBC driver waits before abandoning an attempt to connect to a data source. This value is used as an application-wide default unless the **LoginTimeout** property of the **rdoEnvironment** object is used to override this value.

Syntax

object.**rdoDefaultLoginTimeout** [**=** *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A value that specifies the number of seconds as described in Settings. (Data type is a **Long** expression.)

Settings

The setting for *value* is the number of seconds to wait for a login request to complete before returning a trappable error to the application. A setting of 0 indicates the timeout is disabled and a connection attempt will wait indefinitely.

Remarks

Login requests are made when the **RemoteData** control creates **rdoConnection** objects or when you use the **OpenConnection** method of the **rdoEnvironment** object. The maximum value is dependent on the data source driver. Any value provided over the maximum is set to this maximum value.

The default timeout value, if not specified, is 15 seconds.

Note When you use Data Access Objects (DAO), the LOGINTIMEOUT argument used in the **Connect** property is not a valid argument for ODBC connect strings. Use the **rdoDefaultLoginTimeout** property instead.

See Also

Connect Property
LoginTimeout Property
OpenConnection Method
QueryTimeout Property
rdoEnvironment Object, **rdoEnvironments** Collection
RemoteData Control

rdoDefaultLoginTimeout Property (Remote Data) Applies To

rdoEngine Object

rdoLocaleID Property (Remote Data)

See Also Applies To

Returns or sets a value indicating the locale of the RDO library.

Syntax

object.**rdoLocaleID** [**=** *value*]

The **rdoLocaleID** 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 value that specifies a locale as described in Settings.

Settings

The settings for *value* are:

Constant	Value	Description
rdLocaleSystem	0	System
rdLocaleEnglish	1	English
rdLocaleFrench	2	French
rdLocaleGerman	3	German
rdLocaleItalian	4	Italian
rdLocaleJapanese	5	Japanese
rdLocaleSpanish	6	Spanish
rdLocaleChinese	7	Chinese

Remarks

The locale determines which language is used when generating error messages.   The **rdoLocaleID** defaults to the Windows system locale when the **rdoEngine** is initialized.

You can override the current locale at any time by setting the **rdoLocaleID** to any of the supported values.   If you use an unsupported value, a trappable error occurs.

When the **rdoLocaleID** property is set or changed, RDO loads the appropriate language dynamic-link library (DLL) to show error messages in the correct language.

If the specified language DLL is not present on the user's machine, RDO is set to **rdLocaleEnglish**, which does not require a separate DLL.   When this happens, an informational message is placed in the **rdoErrors** collection indicating that RDO was unable to load the resource DLL for the specified locale.

See Also

rdoEngine Object

rdoError Object, **rdoErrors** Collection

rdoLocaleID Property (Remote Data) Applies To

rdoEngine Object

rdoVersion Property (Remote Data)

See Also Applies To

Returns a value that indicates the version of the RDO library associated with the object.

Syntax

*object***.rdoVersion**

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

Return Values

The **rdoVersion** property return value is a 5-character string expression.

Remarks

For an **rdoEngine** object, this property identifies the version of the database engine that created the connection.   The version is in the form ##.##, where the first two digits are the major version number and the last two digits are the minor version.

See Also

OpenConnection Method
rdoConnection Object, **rdoConnections** Collection
Version Property

rdoVersion Property (Remote Data) Applies To

rdoEngine Object

Resultset Property (Remote Data)

See Also Applies To

Returns or sets an **rdoResultset** object defined by a **RemoteData** control's properties or as returned by the **OpenResultset** method.

Syntax

Set *object*.**Resultset** [= *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	An object expression that evaluates to an **rdoResultset** object as described in Settings.

Settings

The setting for *value* is an **rdoResultset** object.

Remarks

The **RemoteData** control is automatically initialized when your application starts. If the **SQL** property is valid, or if you set the **SQL** property at run time and use the **Refresh** method, the **RemoteData** control attempts to create a new **rdoResultset** object. This **rdoResultset** is accessible through the **RemoteData** control's **Resultset** property.

You can also determine the type of **rdoResultset** cursor to be created by setting or examining the **RemoteData** control's **ResultsetType** property. If you don't request a specific type when using the **RemoteData** control, a keyset-type **rdoResultset** is created. You can determine the type of **rdoResultset** at run time by examining the **rdoResultset** object's **Type** property or the **RemoteData** control's **ResultsetType** property.

The **RemoteData** control can create either keyset- or static-type **rdoResultset** objects when accessing SQL Server 6.0. However, if the ODBC driver does not support keyset cursors, they cannot be created – all drivers support static cursors. A trappable error is triggered if you set the **RemoteData** control's **Resultset** property to an unsupported type of **rdoResultset**.

If you create an **rdoResultset** object using either RDO code or another **RemoteData** control, you can set the **Resultset** property of the **RemoteData** control to this new **rdoResultset**. Any existing **rdoResultset** in the **RemoteData** control, and the **rdoConnection** object associated with it, are released when a new **rdoResultset** is assigned to the **Resultset** property.

Note When the **Resultset** property is set, the **RemoteData** control doesn't close the current **rdoResultset** or **rdoConnection,** but it does release it. If there are no other users, the **rdoConnection** is closed automatically. You may want to consider closing the **rdoResultset** and **rdoConnection** associated with the **RemoteData** control before setting the **Resultset** property.

You can also create an **rdoResultset** object using the **OpenResultset** method and setting the **Resultset** property to the resulting **rdoResultset** object. However, the bound controls using the **RemoteData** control must correctly specify the columns of the new **rdoResultset**. To do so, make sure the **DataField** properties of the bound controls connected to the **RemoteData** control are set to match the new **rdoResultset** object's column names. For example, to create an **rdoResultset** in code and pass it to an existing **RemoteData** control, use code like the following:

```
Public Cn As rdoConnection, Rs As rdoResultset
Sub ApplyrdoResultset()
   Set Cn = rdoEnvironments(0).OpenConnection("MyDSN")
   Set Rs = Cn.OpenResultset("Select * From MyTable")
   Debug.Print Rs.Type                ' Show type created.
   Set RemoteData1.Resultset = Rs     ' Assign rdoResultset.
```

```
End Sub
```

All **rdoResultset** objects created by the **RemoteData** control are built in **rdoEnvironments**(0).   If you need to use the **RemoteData** control to manipulate a database in another **rdoEnvironment**, use the technique demonstrated in the preceding example to open the **rdoConnection** in the desired **rdoEnvironment**, create a new **rdoResultset**, and set the **RemoteData** control's **Resultset** property to this new **rdoResultset**.

Resultset Property (Remote Data) Applies To

RemoteData Control

See Also

MoveFirst, **MoveLast**, **MoveNext**, **MovePrevious** Methods
OpenResultset Method
rdoConnection Object, **rdoConnections** Collection
rdoEnvironment Object, **rdoEnvironments** Collection
rdoResultset Object, **rdoResultsets** Collection
RemoteData Control
ResultsetType Property
RowCount Property
SQL Property
Type Property
Understanding Cursors
Updatable Property

ResultsetType Property (Remote Data)

See Also Applies To

Returns or sets a value indicating the type of **rdoResultset** cursor created or to create.

Syntax

object.**ResultsetType** [= *value*]

The **ResultsetType** 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 value that specifies a type of **rdoResultset**, as described in Settings.

Settings

The settings for *value* are:

Constant	Value	Description
rdOpenStatic	3	(Default) A static-type **rdoResultset**.
rdOpenKeyset	1	A keyset-type **rdoResultset**.

Remarks

Not all drivers support all types of cursors. For example, SQL Server 6.0 supports both static and keyset cursors, but SQL Server 4.2 only supports static cursors. If the ODBC driver does not support keyset cursors, they cannot be created by RDO or the **RemoteData** control. If the **RemoteData** control can't create the type of **rdoResultset** cursor requested, RDO builds one of the types that can be created and returns the cursor type in the **ResultsetType** property.

If you don't specify a **ResultsetType** before the **RemoteData** control creates the **rdoResultset**, a forward-only-type **rdoResultset** is created.

If you create an **rdoResultset** and set the **Resultset** property with this new object, the **ResultsetType** property of the **RemoteData** control is set to the **Type** property of the new **rdoResultset**.

See Also

OpenResultset Method
rdoResultset Object, **rdoResultsets** Collection
Refresh Method
RemoteData Control
Requery Method
Type Property
Understanding Cursors

ResultsetType Property (Remote Data) Applies To

RemoteData Control

ReadOnly Property (Remote Data)

See Also Applies To

Returns or sets a value that determines whether the control's **rdoConnection** is opened for read-only access.

Syntax

object.**ReadOnly** [**=** *value*]

The **ReadOnly** 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 read/write access, as described in Settings.

Settings

The settings for *value* are:

Setting	Description
True	The **RemoteData** control's **rdoConnection** object is opened with read-only access. Changes to data aren't allowed.
False	(Default) The **RemoteData** control's **rdoConnection** is opened with read/write access to data.

Remarks

Use the **ReadOnly** property with a **RemoteData** control to specify whether data in the underlying **rdoConnection** can be changed. For example, you might create an application that only displays data. Accessing a read-only **rdoConnection** might be faster.

Even if the **ReadOnly** property is **False**, a user might not have write access to a database because the user does not have permission or the type of **rdoResultset** in use does not support updates.

This property corresponds to the *readonly* argument in the **OpenConnection** method.

ReadOnly Property (Remote Data) Applies To

RemoteData Control

See Also

Connection Property
OpenConnection Method
rdoConnection Object, **rdoConnections** Collection
Refresh Method
RemoteData Control

Required Property (Remote Data)

See Also Applies To

Returns a value that indicates whether an **rdoColumn** requires a non-**Null** value.

Syntax

*object***.Required**

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

Return Values

The return values for the **Required** property are:

Value	Description
True	A **Null** value isn't allowed.
False	A **Null** value is allowed.

Remarks

For an **rdoColumn** object, you can use the **Required** property along with the **AllowZeroLength** property to determine the validity of the **Value** property setting for that **rdoColumn** object. If **Required** is set to **False**, the column can contain **Null** values as well as values that meet the conditions specified by the **AllowZeroLength** property setting.

See Also

AllowZeroLength Property
rdoColumn Object, **rdoColumns** Collection
rdoPreparedStatement Object, **rdoPreparedStatements** Collection
Value Property

Required Property (Remote Data) Applies To

rdoColumn Object

Restartable Property (Remote Data)

See Also Applies To

Returns a value that indicates whether an **rdoResultset** object supports the **Requery** method, which re-executes the query the **rdoResultset** is based on.

Syntax

*object***.Restartable**

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

Return Values

The **Restartable** property return values are:

Value	Description
True	The **rdoResultset** object supports the **Requery** method.
False	The **rdoResultset** object doesn't support the **Requery** method.

Remarks

Check the **Restartable** property before using the **Requery** method on an **rdoResultset**. If the object's **Restartable** property is set to **False**, use the **OpenResultset** method on the underlying **rdoPreparedStatement** to re-execute the query.

You can use the **Requery** method to update an **rdoResultset** object's underlying parameter query after the parameter values have been changed.

If the **rdoPreparedStatement** does not contain parameters, the **Restartable** property is always **True**.

See Also

OpenResultset Method
rdoPreparedStatement Object, **rdoPreparedStatements** Collection
rdoResultset Object, **rdoResultsets** Collection
Requery Method

Restartable Property (Remote Data) Applies To

rdoResultset Object

RowCount Property (Remote Data)

See Also Applies To

Returns the number of rows accessed in an **rdoResultset** object.

Syntax

*object***.RowCount**

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

Return Values

The **RowCount** property return value is a **Long** integer as discussed in Remarks.

Remarks

Use the **RowCount** property to find out how many rows in an **rdoResultset** object have been accessed. **RowCount** doesn't indicate how many rows will be returned by an **rdoResultset** query. After all rows have been accessed, the **RowCount** property reflects the total number of rows in the **rdoResultset**.

Depending on the driver and data source, the **RowCount** property returns either -1 to indicate that the number of rows is not available, or 0 to indicate that no rows were returned by the **rdoResultset**. If the driver is capable of returning a row count, the **RowCount** property returns the number of rows in the **rdoResultset**.

Using the **Requery** method on an **rdoResultset** resets the **RowCount** property, just as it does when a query is run for the first time.

See Also

MoveFirst, **MoveLast**, **MoveNext**, **MovePrevious** Methods
rdoResultset Object, **rdoResultsets** Collection
rdoTable Object, **rdoTables** Collection
Requery Method

RowCount Property (Remote Data) Applies To

rdoResultset Object
rdoTable Object

RowsAffected Property (Remote Data)

See Also   Applies To

Returns the number of rows affected by the most recently invoked **Execute** method.

Syntax

*object***.RowsAffected**

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

Return Values

The **RowsAffected** property return value is a **Long** value ranging from 0 to the number of rows affected by the most recently invoked **Execute** method on either an **rdoConnection** or **rdoPreparedStatement** object.

Remarks

RowsAffected contains the number of rows deleted, updated, or inserted when running an action query.   When you use the **Execute** method to run an **rdoPreparedStatement**, the **RowsAffected** property setting is the number of rows affected.   For example, when you execute a query that deletes 50 rows from a table, the **RowsAffected** property returns 50.

See Also

Execute Method

rdoPreparedStatement Object, **rdoPreparedStatements** Collection

rdoResultset Object, **rdoResultsets** Collection

RowsAffected Property (Remote Data) Applies To

rdoConnection Object
rdoPreparedStatement Object

RowsetSize Property (Remote Data)

Returns or sets a value that determines the number of rows in an **rdoResultset** cursor.

Syntax

object.**RowsetSize** [**=** *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A value that specifies the size of the rowset as described in Settings. (Data type is a **Long** expression.)

Settings

The upper limit of the **RowsetSize** is determined by the data source driver. The lower limit for *value* is 1, and the default value is 100.

Remarks

The **RowsetSize** property determines how many rows of the keyset are buffered by the application. This property must be set before creating an **rdoResultset** object.

Tuning the size of **RowsetSize** can affect performance and the amount of memory required to maintain the keyset buffer.

See Also

KeysetSize Property
MaxRows Property
Understanding Cursors

RowsetSize Property (Remote Data) Applies To

rdoPreparedStatement Object

Size Property (Remote Data)

See Also Applies To

Returns a value that indicates the maximum size, in bytes, of the underlying data of an **rdoColumn** object that contains text or the fixed size of an **rdoColumn** object that contains text or numeric values.

Syntax

object.**Size**

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

Return Values

The **Size** property return value is a **Long** value. The value depends on the **Type** property setting of the **rdoColumn** object, as discussed in Remarks.

Remarks

For columns that return character values, the **Size** property indicates the maximum number of characters that the data source column can hold. For numeric columns, the **Size** property indicates how many bytes of data source storage are required for the column data. This value depends on the data source implementation.

For data source columns that require the use of **GetChunk** and **AppendChunk** methods, the **Size** property is always 0 –you can use the **ColumnSize** method to return correct size information. The maximum size of a *chunk*-type column is limited only by your system resources or the maximum size of the database.

See Also

Attributes Property
BindThreshold Property
ColumnSize Method
rdoColumn Object, **rdoColumns** Collection
rdoPreparedStatement Object, **rdoPreparedStatements** Collection
rdoResultset Object, **rdoResultsets** Collection
Type Property

Size Property (Remote Data) Applies To

rdoColumn Object

Source Property (Remote Data)

See Also Applies To

Returns a value that indicates the source of a remote data access error.

Syntax

object.**Source**

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

Return Values

The **Source** property return value is a string expression as described in Remarks.

Remarks

When an error occurs during an ODBC operation, an **rdoError** object is appended to the **rdoErrors** collection. If the error occurred within RDO, the return value begins with "MSRDO32". The object class that caused the error might also be appended to the value of the **Source** property.

See Also

Description Property
HelpContext, **HelpFile** Properties
Number Property
rdoError Object, **rdoErrors** Collection
SQLRetCode Property
SQLState Property

Source Property (Remote Data) Applies To

rdoError Object

SourceColumn, SourceTable Properties (Remote Data)

See Also Applies To

– **SourceColumn**
–returns a value that indicates the name of the column that is the original source of the data for an **rdoColumn** object.

– **SourceTable**
–returns a value that indicates the name of the table that is the original source of the data for an **rdoColumn** object.

This property is not available at design time and is read-only at run time.

Syntax

*object***.SourceColumn**

*object***.SourceTable**

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

Return Values

The **SourceColumn** property returns a string expression that specifies the name of the column that is the source of data. The **SourceTable** property returns a string expression that specifies the name of the table that is the source of data.

Remarks

These properties indicate the original column and table names associated with an **rdoColumn** object. For example, you could use these properties to determine the original source of the data in a query column whose name is unrelated to the name of the column in the underlying table.

For columns in **rdoResultset** objects, **SourceColumn** and **SourceTable** return the column name and table name of the base table or the columns and table(s) used to define the query.

See Also

rdoColumn Object, **rdoColumns** Collection
rdoPreparedStatement Object, **rdoPreparedStatements** Collection
rdoResultset Object, **rdoResultsets** Collection
rdoTable Object, **rdoTables** Collection

SourceColumn, SourceTable Properties (Remote Data) Apply To

rdoColumn Object

SQL Property (Remote Data)

See Also Applies To

Returns or sets the SQL statement that defines the query executed by an **rdoPreparedStatement** object or a **RemoteData** control.

Syntax

object.**SQL** [**=** *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A string expression that contains a value as described in Settings. (Data type is **String**.)

Settings

The settings for *value* are:

Setting	Description
A table name	The name of one of the **rdoTable** objects defined in the **rdoConnection** object's **rdoTables** collection.
A valid SQL statement	An SQL query using syntax appropriate for the data source.
An **rdoPreparedStatement**	The name of one of the **rdoPreparedStatement** objects in the **rdoConnection** object's **rdoPreparedStatements** collection.
An **rdoResultset**	The name of one of the **rdoResultset** objects in the **rdoConnection** object's **rdoResultsets** collection.
A stored procedure	The name of a stored procedure supported by the data source preceded with the keyword "execute".

Remarks

The **SQL** property contains the structured query language statement that determines how rows are selected, grouped, and ordered when you execute a query. You can use a query to select rows to include in an **rdoResultset** object. You can also define action queries to modify data without returning rows.

Note You can't use the **rdoTable** object names until the **rdoTables** collection is referenced. When your code references the **rdoTables** collection by enumerating one or more of its members, RDO queries the data source for table meta data. This results in population of the **rdoTables** collection. This means that you cannot simply provide a table name for the *value* argument without first enumerating the **rdoTables** collection.

The SQL syntax used in a query must conform to the SQL dialect as defined by the data source query processor. The SQL dialect supported by the ODBC interface is defined by the X/Open standard. Generally, a driver scans an SQL statement looking for specific escape sequences that are used to identify non-standard operands like timestamp literals and functions.

If the SQL statement includes parameter declarations for the query, you must set these before you execute the query. Until you reset the parameters, the same parameter values are applied each time you execute the query. To use the **rdoParameters** collection to manage SQL query parameters, you must include the "?" parameter placeholder in the SQL statement. Input, output, and return value parameters must all be identified in this manner. Use the **Direction** property to indicate how the parameter will be used. For example, to execute a procedure that accepts two input parameters and returns a return value and an output parameter, you can use the following code:

```
QSQL$ = "{ ? = call sp_MyProc (?, ?, ?) }"
Set CPw = cn.CreatePreparedStatement("",QSQL$)
Cpw.rdoParameters(0).Direction = rdReturnValue
Cpw.rdoParameters(1).Direction = rdParamInput
Cpw.rdoParameters(2).Direction = rdParamInput
Cpw.rdoParameters(3).Direction = rdParamOutput
Set MyRs = Cpw.OpenResultSet()
```

Note When using Microsoft SQL Server 6.0 as a data source, the ODBC driver automatically sets the **Direction** property. You also do not need to set the **Direction** property for input parameters, as this is the default setting.

If the user changes the parameter value, you can re-apply the parameter value and re-execute the query by using the **Requery** method against the **rdoResultset** (`MyRs`).

```
Cpw.rdoParameters(0) = Text1.Text
MyRs.Requery
```

You can also specify parameters in any SQL statement by concatenating the parameters to the SQL statement string. For example, to execute the previous example using this technique, you can use the following code:

```
QSQL$ = "SELECT * FROM Authors WHERE Au_Lname = '"
QSQL$ = QSQL$ & Text.Text & "'"
Set CPw = cn.CreatePreparedStatement("",QSQL$)
Set MyRs = Cpw.OpenResultSet()
```

In this case, the **rdoParameters** collection is not created and cannot be referenced. To change the query parameter, you must rebuild the SQL statement with the new parameter value each time the query is executed, or before you use the **Requery** method.

The SQL statement may include an ORDER BY clause to change the order of the rows returned by the **rdoResultset** or a WHERE clause to filter the rows.

RemoteData Control

When used with the **RemoteData** control, the **SQL** property specifies the source of the data rows accessible through bound controls on your form.

If you set the **SQL** property to an SQL statement that returns rows or to the name of an existing **rdoPreparedStatement**, all columns returned by the **rdoResultset** are visible to the bound controls associated with the **RemoteData** control.

After changing the value of the **SQL** property at run time, you must use the **Refresh** method to activate the change.

Note Whenever your **rdoPreparedStatement** or SQL statement returns a value from an expression, the column name of the expression is determined by the wording of the SQL query. In most cases you'll want to alias expressions so you know the name of the column to bind to the bound control.

Make sure each bound control has a valid setting for its **DataField** property. If you change the setting of a **RemoteData** control's **SQL** property and then use **Refresh**, the **rdoResultset** identifies the new object. This may invalidate the **DataField** settings of bound controls and cause a trappable error.

See Also

CreatePreparedStatement Method
Creating Parameter Queries
Execute Method
OpenResultset Method
rdoParameter Object, **rdoParameters** Collection
rdoPreparedStatement Object, **rdoPreparedStatements** Collection
rdoResultset Object, **rdoResultsets** Collection
rdoTable Object, **rdoTables** Collection

SQL Property (Remote Data) Applies To

RemoteData Control
rdoPreparedStatement Object

SQLRetCode Property (Remote Data)

See Also Applies To

Returns the error return code from the most recent RDO operation.

Syntax

object.**SQLRetCode**

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

Return Values

The **SQLRetCode** property return value is a **Long** value that corresponds to one of the following constants:

Constant	Description
rdSQLSuccess	The operation is successful.
rdSQLSuccessWithInfo	The operation is successful, and additional information is available.
rdSQLNoDataFound	No additional data is available.
rdSQLError	An error occurred performing the operation.
rdSQLInvalidHandle	The handle supplied is invalid.

Remarks

The **SQLRetCode** property contains the ODBC return code for the error.

See Also

Description Property
HelpContext, **HelpFile** Properties
Number Property
rdoError Object, **rdoErrors** Collection
Source Property
SQLState Property

SQLRetCode Property (Remote Data) Applies To

rdoError Object

SQLState Property (Remote Data)

See Also Applies To

Returns a value corresponding to the type of error as defined by the X/Open and SQL Access Group SQL.

Syntax

object.**SQLState**

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

Return Values

The **SQLState** return value is a five-character string expression, as described in Remarks.

Remarks

When an RDO operation returns an error, or completes an operation, the **SQLState** property of the **rdoError** object is set. If the error is not caused by ODBC or if no **SQLState** is available, the **SQLState** property returns an empty string.

The character string value returned by the **SQLState** property consists of a two-character class value followed by a three-character subclass value. A class value of "01" indicates a warning and is accompanied by a return code of **rdSQLSuccessWithInfo**.

Class values other than "01", except for the class "IM", indicate an error and are accompanied by a return code of **rdSQLError**. The class "IM" is specific to warnings and errors that derive from the implementation of ODBC itself. The subclass "000" in any class is for implementation-defined conditions within the given class. The assignment of class and subclass values is defined by ANSI SQL-92.

See Also

Description Property
HelpContext, **HelpFile** Properties
Number Property
rdoError Object, **rdoErrors** Collection
Source Property
SQLRetCode Property

SQLState Property (Remote Data) Applies To

rdoError Object

StillExecuting Property (Remote Data)

See Also Applies To

Returns a Boolean value that indicates whether a query is still executing.

Syntax

object.**StillExecuting**

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

Return Values

The **StillExecuting** property return values are:

Value	Description
True	The asynchronous query is still executing.
False	The asynchronous query is ready to return the first result set.

Remarks

Use the **StillExecuting** property to determine if a query executed using the **rdAsyncEnable** option is ready to return the first result set. Until the **StillExecuting** property is **False**, the associated object cannot be accessed.

Once the **StillExecuting** property returns **False**, the first or next result set is ready for processing. When you use the **MoreResults** method to complete processing of a result set, the **StillExecuting** property is reset to **True** while subsequent result sets are retrieved.

Use the **Cancel** method to terminate processing of an executing query, including all statements in a batch query.

See Also

Cancel Method
Execute Method
MoreResults Method
OpenResultset Method

StillExecuting Property (Remote Data) Applies To

rdoConnection Object
rdoPreparedStatement Object
RemoteData Control
rdoResultset Object

Transactions Property (Remote Data)

See Also Applies To

Returns a value that indicates whether an object supports the recording of a series of changes that can later be rolled back (undone) or committed (saved).

Syntax

*object***.Transactions**

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

Return Values

The **Transactions** property return values are:

Value	Description
True	The object supports transactions.
False	The object doesn't support transactions.

Remarks

Check the **Transactions** property before using the **BeginTrans** method to make sure that transactions are supported. When **Transactions** is **False,** using the **BeginTrans**, **CommitTrans**, or **RollbackTrans** method has no effect.

See Also

BeginTrans, **CommitTrans**, **RollbackTrans** Methods
rdoConnection Object, **rdoConnections** Collection
rdoEnvironment Object, **rdoEnvironments** Collection
rdoResultset Object, **rdoResultsets** Collection
rdoTable Object, **rdoTables** Collection

Transactions Property (Remote Data) Applies To

rdoConnection Object
rdoResultset Object

Type Property (Remote Data)

See Also   Applies To

Returns a value that indicates the type or data type of an object.

Syntax

object.**Type**

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

Return Values

For an **rdoColumn** or **rdoParameter** object, the **Type** property returns an **Integer**.   The return values are:

Constant	Value	Description
rdTypeCHAR	1	Fixed-length character string. Length set by **Size** property.
rdTypeNUMERIC	2	Signed, exact, numeric value with precision p and scale s (1 p 15; 0 s p).
rdTypeDECIMAL	3	Signed, exact, numeric value with precision p and scale s (1 p 15; 0 s p).
rdTypeINTEGER	4	Signed, exact numeric value with precision 10, scale 0 (signed: -231 n 231-1; unsigned: 0 n 232-1).
rdTypeSMALLINT	5	Signed, exact numeric value with precision 5, scale 0 (signed: -32,768 n 32,767, unsigned: 0 n 65,535).
rdTypeFLOAT	6	Signed, approximate numeric value with mantissa precision 15 (zero or absolute value 10-308 to 10308).
rdTypeREAL	7	Signed, approximate numeric value with mantissa precision 7 (zero or absolute value 10-38 to 1038).
rdTypeDOUBLE	8	Signed, approximate numeric value with mantissa precision 15 (zero or absolute value 10-308 to 10308).
rdTypeDATE	9	Date – data source dependent.
rdTypeTIME	10	Time – data source dependent.
rdTypeTIMESTAMP	11	TimeStamp – data source dependent.
rdTypeVARCHAR	12	Variable-length character string. Maximum length 255.
rdTypeLONGVARCHAR	-1	Variable-length character string. Maximum length determined by data source.
rdTypeBINARY	-2	Fixed-length binary data. Maximum length 255.
rdTypeVARBINARY	-3	Variable-length binary data. Maximum length 255.
rdTypeLONGVARBINARY	-4	Variable-length binary data. Maximum data source dependent.
rdTypeBIGINT	-5	Signed, exact numeric value with precision 19 (signed) or 20 (unsigned), scale 0; (signed: -263 n 263-1; unsigned: 0 n 264-1).
rdTypeTINYINT	-6	Signed, exact numeric value with precision 3, scale 0; (signed: -128 n 127, unsigned: 0 n 255).
rdTypeBIT	-7	Single binary digit.

For an **rdoPreparedStatement** object, the **Type** property returns an **Integer**.   The return values are:

Constant	Value	Query type

rdQSelect	0	Select
rdQAction	1	Action
rdQProcedure	2	Procedural

For an **rdoResultset** object, the **Type** property returns an **Integer**. The return values are:

Constant	Value	rdoResultset type
rdOpenForwardOnly	0	Fixed set, non-scrolling.
rdOpenKeyset	1	Updatable, fixed set, scrollable query result set cursor.
rdOpenDynamic	2	Updatable, dynamic set, scrollable query result set cursor.
rdOpenStatic	3	Read-only, fixed set.

Note Not all ODBC drivers or data sources support every type of cursor. If you choose a type of cursor that is not supported, the ODBC driver reverts to a supported type.

For an **rdoTable** object, the **Type** property returns a **String**. The settings for *value* are determined by the data source driver.

Typically, this string value is "TABLE", "VIEW", "SYSTEM TABLE", "GLOBAL TEMPORARY". "LOCAL TEMPORARY", "ALIAS", "SYNONYM" or some other data source-specific type identifier.

Remarks

Depending on the object, the **Type** property indicates:

Object	Type indicates
rdoColumn, rdoParameter	Object data type
rdoPreparedStatement	Type of query
rdoResultset	Type of **rdoResultset**
rdoTable	Type of table on data source

See Also

rdoColumn Object, **rdoColumns** Collection
rdoParameter Object, **rdoParameters** Collection
rdoPreparedStatement Object, **rdoPreparedStatements** Collection
rdoResultset Object, **rdoResultsets** Collection
rdoTable Object

Type Property (Remote Data) Applies To

rdoColumn Object
rdoParameter Object
rdoPreparedStatement Object
rdoResultset Object
rdoTable Object

Updatable Property (Remote Data)

See Also Applies To

Returns a Boolean value that indicates whether changes can be made to a remote data object.

Syntax

object.**Updatable**

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

Return Values

The **Updatable** property return values are:

Value	Description
True	The object can be changed or updated.
False	The object can't be changed or updated. This is the only setting for static-type **rdoResultset** objects.

Remarks

If the **Updatable** property setting is **True**, the specified:

- **rdoConnection** object refers to an updatable data source.
- **rdoPreparedStatement** object refers to an updatable result set.
- **rdoResultset** contains updatable rows.
- **rdoTable** object refers to a table that can be changed.
- **rdoColumn** object refers to data that can be changed.

You can use the **Updatable** property with all types of **rdoResultset** objects.

Many types of **rdoResultset** objects can contain columns that can't be updated. For example, you can create a forward-only **rdoResultset** that is derived from nonupdatable sources or that contains computed or derived columns.

If the object contains only nonupdatable columns, the value of the **Updatable** property is **False**. When one or more columns are updatable, the property's value is **True**. You can edit only the updatable columns. A trappable error occurs if you try to assign a new value to a nonupdatable column.

Because an updatable object can contain columns that cannot be updated, check the **Updatable** property of each **rdoColumn** before editing a row in the **rdoResultset**.

See Also

rdoColumn Object, **rdoColumns** Collection
rdoPreparedStatement Object, **rdoPreparedStatements** Collection
rdoResultset Object, **rdoResultsets** Collection
rdoTable Object, **rdoTables** Collection
Update Method

Updatable Property (Remote Data) Applies To

rdoColumn Object
rdoConnection Object
rdoPreparedStatement Object
rdoResultset Object
rdoTable Object

UserName Property (Remote Data)

See Also Applies To

Returns or sets a value that represents a user of an **rdoEnvironment** object. Use the **UserName** property with the **Password** property to connect to an ODBC data source.

Syntax

*object***.UserName** [**=** *value*]

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

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A string expression that contains a user name as described in Settings. (Data type is **String**.)

Settings

The user name syntax depends on the ODBC data source.

Return Values

The **UserName** property represents the user of an **rdoEnvironment** object. The user name is set when the **rdoEnvironment** is either created automatically by the **RemoteData** control, by the first reference to a remote data object, or when the **rdoCreateEnvironment** method is executed.

You can determine the default user name by setting or examining the **rdoDefaultUser** property of the **rdoEngine** object. If no specific user name is supplied in **UserName**, the value of the **rdoDefaultUser** property is used.

See Also

Password Property
rdoCreateEnvironment Method
rdoDefaultUser, **rdoDefaultPassword** Properties
rdoEnvironment Object, **rdoEnvironments** Collection Summary

UserName Property (Remote Data) Applies To

rdoEnvironment Object

Value Property (Remote Data)

See Also Applies To

Returns or sets the value of an object.

Syntax

object.**Value** [**=** *value*]

The **Value** 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 appropriate for the data type, as specified by the **Type** property of an object. (Data type is **Variant**.)

Remarks

Use the **Value** property to retrieve and alter data in **rdoResultset** objects.

The **Value** property is the default property of the **rdoColumn** and **rdoParameter** objects. Therefore, the following lines of code are equivalent (assuming Column1 is at the first ordinal position):

```
Dim MyResultset As rdoResultset
X = MyResultset!Column1
X = MyResultset!Column1.Value
X = MyResultset(0)
X = MyResultset(0).Value
X = MyResultset("Column1").Value
X = MyResultset("Column1")
X = RemoteData1.Resultset("Column1")
X = RemoteData1.Resultset(0)
F$ = "Column1" : X = MyResultset(F$).Value
X = MyResultset(F$)
Set X = MyResultset(0): X.Value : X
```

See Also

Execute Method
Name Property
OpenResultset Method
rdoColumn Object, **rdoColumns** Collection
rdoParameter Object, **rdoParameters** Collection
Type Property
Updatable Property

Value Property (Remote Data) Applies To

rdoColumn Object
rdoParameter Object

Version Property (Remote Data)

See Also Applies To

Returns a value that indicates the version of the data source associated with the object.

Syntax

object.**Version**

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

Return Values

The **Version** property return value is a 10-character string expression.

Remarks

For an **rdoConnection** object, this property identifies the version of the data source used when the connection was created. This value is the version of ODBC to which the driver manager conforms. The version is in the form ##.##.####, where the first two digits are the major version number, the next two digits are the minor version, and the last four digits are the build number.

See Also

OpenConnection Method
rdoConnection Object, **rdoConnections** Collection
rdoVersion Property

Version Property (Remote Data) Applies To

rdoConnection Object

Already beyond the end of the result set (Error 40024)

See Also

You attempted to call **rdoResultset.MoveNext** when the **EOF** property was set to **True**. To avoid this error, check the state of the **EOF** property before calling **MoveNext**.

See Also

BOF, **EOF** Properties

MoveFirst, **MoveLast**, **MoveNext**, **MovePrevious** Methods

An error occurred configuring the DSN. Please check the parameters and try again (Error 40000)

See Also

The call to **rdoEngine.rdoRegisterDataSource** failed.

To avoid this error, check the validity of the **rdoRegisterDataSource** arguments passed and try again.

See Also

rdoRegisterDataSource Method

An error occurred loading the ODBC installation library (ODBCCP32.DLL) (Error 40032)

See Also

You attempted to call **rdoEngine.rdoRegisterDataSource** and the application could not load the ODBC installation library file ODBCCP32.DLL.

To avoid this error, make sure ODBC is correctly installed on the machine that generated the error, and that the file ODBCCP32.DLL is in the system path.

See Also

rdoRegisterDataSource Method

An internal ODBC error was encountered (Error 40002)

See Also

An ODBC error occurred on the most recently invoked property or method.

The exact error depends on the ODBC driver and type of database you are using. Examine the **rdoErrors** Collection for an exact description of the problem.

Note ODBC can generate more than one error during statement execution. Make sure you check each error in the **rdoErrors** collection.

See Also

rdoError Object, **rdoErrors** Collection

An invalid ODBC handle was encountered (Error 40004)

See Also

An error caused by an invalid ODBC statement handle occurred when executing an ODBC operation.

Examine the **rdoErrors** collection for an exact description of the problem.   If no information is found, make sure the statement handle wasn't deallocated or altered by a previous operation.

See Also

rdoError Object, **rdoErrors** Collection

An invalid value for the concurrency option was passed (Error 40019)

See Also

An invalid lock type was passed to either **rdoPreparedStatement.LockType** or the *locktype* argument in **rdoPreparedStatement.OpenResultset**.

To avoid this error, make sure you pass one of the following valid lock types:

- **rdConcurReadOnly**
- **rdConcurLock**
- **rdConcurRowver**
- **rdConcurValues**

Note Not all lock types can be used on every data source.

See Also

LockType Property

OpenResultset Method

An invalid value for the cursor driver was passed (Error 40003)

See Also

An invalid type value was passed to either **rdoEnvironment.CursorDriver** or **rdoEngine.rdoDefaultCursorDriver**.

To avoid this error, pass one of the following values:

- **rdUseIfNeeded**
- **rdUseODBC**
- **rdUseServer**

Note Not all data source drivers support all cursors.

See Also

CursorDriver Property

rdoDefaultCursorDriver Property

An invalid value for the prompt option was passed (Error 40033)

See Also

You attempted to call **rdoEnvironment.OpenConnection**, and the value for the *prompt* argument was not one of the following values:

- **rdDriverPrompt**
- **rdDriverNoPrompt**
- **rdDriverComplete**
- **rdDriverCompleteRequired**

To avoid this error, make sure the *prompt* argument is one of the previously mentioned values.

See Also

OpenConnection Method

An invalid value for the cursor type parameter was passed (Error 40034)

See Also

You attempted to call the **OpenResultset** method, and the value for the *type* argument was not one of the following values:

- **rdOpenKeyset**
- **rdOpenDynamic**
- **rdOpenStatic**
- **rdOpenForwardOnly**

To avoid this error, make sure the *type* argument is one of the previously mentioned values.

Note Not all data sources support all cursors.

See Also

OpenResultset Method

Type Property

BOF already set   (Error 40025)

See Also

You attempted to call **rdoResultset.MovePrevious** when the **BOF** property was set to **True**.   This means that the current row pointer is already positioned before the first row in the result set, and you are trying to perform a move operation that would move the row pointer to an invalid position.

To avoid this error, check the state of the **BOF** property before calling the **MovePrevious** method.

See Also

BOF, **EOF** Properties

MoveFirst, **MoveLast**, **MoveNext**, **MovePrevious** Methods

Can't create prepared statement for invalid database connection (Error 40015)

See Also

The program tried to create an **rdoPreparedStatement** on an invalid **rdoConnection**.

To avoid this problem, make sure the **rdoConnection** object is currently connected to a data source.

See Also

CreatePreparedStatement Method
rdoConnection Object, **rdoConnections** Collection
rdoPreparedStatement Object, **rdoPreparedStatements** Collection

Can't execute empty rdoPreparedStatement (Error 40018)

See Also

A Null or empty string was encountered in the **SQL** property of **rdoPreparedStatement** during the invocation of the **Execute** method.

To avoid this error, make sure the SQL statement for the **rdoPreparedStatement** is valid, either by passing it as an argument to **CreatePreparedStatement**, or by setting the **SQL** property of the **rdoPreparedStatement** object.

See Also

CreatePreparedStatement Method

Execute Method

SQL Property

Can't execute unprepared rdoPreparedStatement (Error 40017)

See Also

An ODBC error occurred trying to prepare the SQL statement passed in **rdoPreparedStatement.Execute**.

Check the **rdoErrors** collection for more detail and make sure the SQL statement for the **rdoPreparedStatement** is valid for the data source you are referencing.

See Also

Execute Method
rdoError Object, **rdoErrors** Collection
rdoPreparedStatement Object, **rdoPreparedStatements** Collection

Can't move relative to current row as EOF/BOF already set (Error 40029)

See Also

You attempted to call **rdoResultset.Move** specifying a relative move (by passing Null as the *bookmark* argument) when the result set is currently positioned at **EOF** or **BOF**, or the result set is marked as invalid.

Note If **EOF** or **BOF** are set, a valid bookmark must be passed as part of the call to **Move**.

To avoid this error, make sure the **EOF** or **BOF** properties are not set, or a valid bookmark is provided.

See Also

BOF, **EOF** Properties
Move Method
rdoResultset Object, **rdoResultsets** Collection

Invalid bookmark (Error 40027)

See Also

An invalid bookmark value was passed to **rdoResultset.Move**.

To avoid this error, be sure to pass a valid bookmark.   Retrieve the bookmark by using **rdoResultset.Bookmark**, and make sure the variable you use to store the bookmark is still valid.

See Also

Bookmark Property
Move Method
rdoResultset Object, **rdoResultsets** Collection

Invalid bookmark argument to move (Error 40028)

See Also

The *bookmark* argument to **rdoResultset.Move** was not passed as the correct data type.

To avoid this error, be sure to pass the *bookmark* argument as either an **Integer** or **Byte** data type. Retrieve the bookmark by using **rdoResultset.Bookmark**, and be sure that the variable you use to store the bookmark is still valid.

See Also

Bookmark Property
Move Method
rdoResultset Object, **rdoResultsets** Collection

Invalid connection string (Error 40005)

See Also

An invalid connection string was passed to **rdoEnvironment.OpenConnection**.

To avoid this error, be sure to pass a valid ODBC connection string to the **OpenConnection** method of **rdoEnvironment**.

See Also

OpenConnection Method

rdoEnvironment Object, **rdoEnvironments** Collection

Invalid resultset state for update (Error 40026)

See Also

You attempted to call **rdoResultset.Update** when the current row pointer was not pointing to a valid row, or the result set was marked as invalid.   This occurs if:

- The current row pointer is pointing at **EOF** or **BOF**.
- The result set has been marked invalid due to a call to a method such as **Cancel**.
- An SQL error occurs.

Also, deleting a row will mark it as invalid.

To avoid this error, check the state of the **BOF** and **EOF** properties before calling **Update**, and make sure no method was called prior to calling **Update** that would mark the result set as invalid.

See Also

Bookmark Property
rdoResultset Object, **rdoResultsets** Collection
Update Method

Invalid state for Move (Error 40023)

See Also

An attempt was made to call either **rdoResultset.MoveFirst** or **rdoResultset.MoveNext** when the result set was marked as invalid. A result set can be marked as invalid if:

- You called the **Cancel** method on the result set prior to calling a **Move** method.
- You called the **MoreResults** method and there are no more result sets.
- An SQL error occurs.

To avoid this error, be sure that the current result set is valid and that you have not called an operation that would mark it as invalid.

See Also

MoreResults Method

MoveFirst, **MoveLast**, **MoveNext**, **MovePrevious** Methods

Object collection: illegal modification -- collection is read-only (Error 40049)

See Also

You attempted to programmatically modify the contents of an RDO collection.   All RDO collections are read-only, with the exception of the **rdoErrors** collection, which supports the **Clear** method.

To avoid this error, do not attempt to modify the contents of the RDO collections.   Items are added to the collection automatically, and they are removed when the **Close** method for an object is invoked.

See Also

Remote Data Objects and Collections

SQL returned No Data Found (Error 40001)

An SQL statement you tried to execute returned a message indicating no data was found for a query, or no rows were affected by the action (**Insert**, **Update**, **Delete**) statement you attempted to execute.   This may be a valid response, however, in cases where you issue a query expecting no data or rows to exist for that query.

To avoid this error, change the criteria in your SQL statement and try again.

An invalid parameter was passed (Error 40054)

The value passed to a property or method is not a valid value.

Most properties and methods accept values of only a certain type, within a certain range, and an inappropriate value has been assigned to a property or method. See the property or method's Help topic to determine the appropriate types and range of values.

Can't assign value to column unless in edit mode (Error 40039)

See Also

You attempted to assign a value to a column before calling **rdoResultset.Edit** or **rdoResultset.AddNew**.

To avoid this error, make sure the **Edit** or **AddNew** method has been called before assigning values to columns with either the **Value** property or the **AppendChunk** method.

See Also

AddNew Method
AppendChunk Method
Edit Method
Value Property

Can't assign value to non-updatable column (Error 40038)

See Also

You attempted to assign a value to a column that is not updatable.

To avoid this error, make sure the **rdoColumn**'s **Updatable** property is **True** before assigning a value to a column with either the **Value** property or the **AppendChunk** method.

See Also

AppendChunk Method
Updatable Property
Value Property

Can't assign value to output-only parameter (Error 40043)

See Also

An attempt was made to set a value for a parameter that is an output parameter.

To avoid this error, use the **Direction** property to be sure the parameter for which you are setting a value is defined as either an input (**rdParamInput**) or input/output (**rdParamInputOutput**) parameter.

See Also

Direction Property

rdoParameter Object, **rdoParameters** Collection

Can't assign value to unbound column (Error 40037)

See Also

You attempted to assign a value to a column that represents a large binary object (BLOB) or similar object.

To avoid this error, use the **AppendChunk** method to assign values to columns of this type.   Use the **ChunkRequired** property to determine if the column in question requires the use of **AppendChunk**.

See Also

AppendChunk Method
BindThreshold Property
ChunkRequired Property
Type Property

Can't assign value to unbound parameter (Error 40042)

An attempt was made to set a value for a parameter that has not been bound.

To avoid this error, make sure no error was returned from prior RDO methods. If an error was returned, fix the SQL statement that was passed, and try the operation again.

Column not bound correctly (Error 40035)

See Also

An attempt was made to open a result set with a column of unknown data type.

For more information, consult the documentation for the data source from which the column data originated.

See Also

OpenResultset Method

Type Property

GetNewEnum: Couldn't get interface for IID_IUnknown (Error 40050)

An internal error occurred while attempting to allocate memory to enable the **For...Each** syntax to iterate over a collection.

Close as many open applications as possible and try the operation again.

Incorrect type for parameter (Error 40040)

See Also

A **Variant** with an invalid type was detected.   This can be caused by passing a value to an RDO method or property that cannot be coerced to the correct type, such as trying to pass a string data type as a numeric value.

To avoid this problem, make sure the value passed is the correct type for the operation. For column values, you can check the column's **Type** property to ensure you are passing the correct type.

See Also

Type Property

Object Collection: Couldn't find item indicated by text (Error 40041)

See Also

You attempted to address an object in a collection by using a text value, and no object in that collection matched the text string supplied.

To avoid this error, make sure an object in the collection has a **Name** property that matches the text string supplied, or use the object's ordinal value.

See Also

Name Property

Object Collection: This collection doesn't support location by text tag (Error 40021)

You attempted to find an object in a collection with a text string, and the objects in the collection do not support lookup by text strings.

To avoid this error, use an ordinal value instead, such as **rdoErrors**(1).

The object has already been closed (Error 40046)

See Also

You have attempted to call the **Close** method on an object that has already been closed.

To avoid this problem, do not use the **Close** method on an object that has already been closed.

See Also

Close Method

This environment name already exists in the collection (Error 40048)

See Also

You attempted to call the **rdoEngine.rdoCreateEnvironment** method or the **rdoConnection.CreatePreparedStatement** method passing a name that already exists in the **rdoEnvironments** or **rdoPreparedStatements** collection.

To avoid this error, make sure the name you pass does not conflict with a name already added to the collection.

See Also

CreatePreparedStatement Method
Name Property
rdoCreateEnvironment Method
rdoEnvironment Object, **rdoEnvironments** Collection
rdoPreparedStatement Object, **rdoPreparedStatements** Collection

Unbound column - use GetChunk (Error 40036)

See Also

You attempted to access a column containing a large binary object (BLOB) or similar object.

To avoid this error, use the **GetChunk** method to access columns of this type. If this error occurs when using parameters, you cannot use parameters on columns that represent large binary objects (Text or Image columns in SQL Server). You can use the **ChunkRequired** property to determine if the column in question requires the use of **GetChunk**.

See Also

BindThreshold Property
ChunkRequired Property
GetChunk Method
Type Property

You cannot execute a query when an asynchronous query is in progress (Error 40045)

See Also

You have attempted to call a method or property while an asynchronous query is still executing.

To avoid this error, check the **StillExecuting** property, and do not call RDO methods or properties that affect the SQL statement until the **StillExecuting** property returns **False**.

See Also

Running Asynchronous Queries

StillExecuting Property

You must specify a valid name for the environment (Error 40047)

See Also

You have attempted to pass an invalid name for the *name* argument when calling the **rdoEngine.rdoCreateEnvironment** method.

To avoid this error, make sure you pass a valid name as the *name* argument.   The name can be any string expression that is not null or empty, and it should not be a duplicate of any name previously added to the collection.

See Also

Name Property
rdoCreateEnvironment Method
rdoEnvironment Object, **rdoEnvironments** Collection

An attempt was made to issue a select statement using the Execute method (Error 40057)

See Also

You attempted to issue a select statement using the **Execute**.

To issue a select query, you should instead use the **OpenResultset** method.   The **Execute** method is designed for use with action queries (**Insert**, **Update**, **Delete**).

See Also

Execute Method
OpenResultset Method
rdoConnection Object, **rdoConnections** Collection
rdoPreparedStatement Object, **rdoPreparedStatements** Collection
rdoResultset Object, **rdoResultsets** Collection

An error occurred loading the version library (VERSION.DLL) (Error 40016)

See Also

You attempted to call **rdoEngine.rdoVersion** and the application could not load the Win32 library file VERSION.DLL.

To avoid this error, make sure the file VERSION.DLL is in the system path.

See Also

rdoVersion Property

An unexpected error occurred (Error 40006)

An unexpected error occurred that caused the RDO to become unstable.

To avoid this error, make sure you have enough free resources and memory, then restart the program and try again.

Incompatible data types for compare (Error 40014)

An **rdoResultset** was called with an argument value whose data type is not compatible with the compared column's data type.

To avoid this problem, make sure the value you are using in the comparison matches the data type of the column you are comparing against.   Also, this method is valid only when called by a data source control.

Invalid operation (Error 40055)

See Also

The property or method invoked is not valid in this context.

To avoid this error, check the sequence of the operations you are attempting and make sure they are correct.   One possible cause is that you are trying to set a column value on a column that is a meta data column (that is, a column generated from an **rdoTable** and not an **rdoResultset**).

See Also

rdoResultset Object, **rdoResultsets** Collection

rdoTable Object, **rdoTables** Collection

Invalid operation for forward-only cursor (Error 40008)

See Also

The program called **rdoResultset.MovePrevious** or **rdoResultset.MoveFirst** while processing a forward-only query.

To avoid this error, either change the cursor type to **rdOpenKeyset**, **rdOpenDynamic**, or **rdOpenStatic**, or call only **MoveNext** for a forward-only **rdoResultset**.

See Also

Move Method

Invalid row for AddNew (Error 40010)

See Also

There was a call to **rdoResultset.AddNew** when the cursor was positioned on an invalid row, or the program had previously called **AddNew** or **Edit** without calling **Update** or cancelling the operation.

To avoid this error, move the cursor to a valid row and make sure you have a valid result set, and the sequence of calls to **rdoResultset** are correct.

See Also

AddNew Method
Edit Method
Update Method

Invalid seek flag (Error 40012)

An invalid seek flag was passed to **rdoResultset**. This message is applicable to developers creating third-party bound controls.

To avoid this error, make sure the flag passed is one of the following values:

- **DBSEEK_LT**
- **DBSEEK_LE**
- **DBSEEK_EQ**
- **DBSEEK_GE**
- **DBSEEK_GT**
- **DBSEEK_PARTIALEQ**

Also, this method is only valid when called by a data source control.

No current row (Error 40009)

See Also

There was a call to **rdoResultset.Edit** when the cursor was positioned on an invalid row. This error may be caused by attempting to edit a deleted row, or by invoking **Edit** when the cursor is positioned either before the first row, or after the last row.

To avoid this error, move the cursor to a valid row and make sure you have a valid result set.

See Also

Edit Method

Object is invalid or not set (Error 40011)

The program called a method or operation on an object that has been closed, discarded, or not allocated.

To avoid this error, make sure:

- The object has been allocated using the **Set** syntax.
- The object or its parent objects have not been closed.
- The object has not been set to **Nothing**.

Partial equality requires string column (Error 40013)

rdoResultset.FindByValues was called specifying **DBSEEK_PARTIALEQ** on a non-string column.   **DBSEEK_PARTIALEQ** works only on columns that contain string data.

To avoid this error, use only **DBSEEK_PARTIALEQ** on string-based columns.   Also, this method is valid only when called by a data source control.

This message is applicable to developers creating third-party bound controls.

The row you attempted to move to has been deleted (Error 40056)

See Also

The row you attempted to move to using a bookmark has been deleted from the database. To avoid this error, try the operation again, specifying a valid bookmark.

See Also

Bookmark Property

The rdoResultset is empty (Error 40022)

See Also

You attempted to make a call to **rdoResultset.Move**, **rdoResultset.MoveNext**, or **rdoResultset.MovePrevious** on an empty result set.

To avoid this error, make sure the SQL statement used returns a valid result set before using any of the previously mentioned methods.   You can check to see if a result set is empty by checking the **RowCount** property, or by checking to see if both the **EOF** and **BOF** properties are **True**.

See Also

BOF, **EOF** Properties
Move Method
RowCount Property

The resultset is read only (Error 40058)

You attempted an **Edit**, **Delete**, or **Add** operation on a read-only result set. Make sure you specify the correct **LockType** value that supports action queries when you open the result set.

The user canceled the operation. (Error 40059)

The user clicked the Cancel button on an ODBC dialog box.

A control canceled the operation or an unexpected internal error has occurred (Error 40503)

The **RemoteData** control tried to update a row based on bound control data, but the **Update** operation failed.   This usually occurs because of one of the following reasons:

- The data in the bound control fails validation.
- The data is not the correct data type for the result set column.
- The value in the bound control does not match the row description, rule, or trigger criteria.

Examine and modify the data values and retry the operation.

An error has occurred. Unable to retrieve error information (Error 40502)

An error has occurred in the **RemoteData** control, and no detailed error information is available.   This error occurs only under very unusual circumstances−never during normal operation.   You'll get this error message only when the **RemoteData** control can't access detailed error information.   This situation only occurs when OLE is not working properly, and is generally an indication of a more serious error condition.

An unexpected error occurred (Error 40501)

The **RemoteData** control tried to call an RDO method that should normally exist. The method was not found, or the call did not complete correctly. This error occurs only under very unusual circumstances–never during normal operation. You'll see this error message only when the **RemoteData** control can't get a dispatch interface to RDO, or when a method in RDO fails for an unknown reason. This error message is generally an indication of a more serious error condition.

An unexpected internal error has occurred (Error 40500)

The **RemoteData** control attempted to notify the bound controls that new data is available, but received a failure code from RDO instead.   This error occurs only under very unusual circumstances–never during normal operation
–and only if something is seriously wrong with Visual Basic's binding manager or the bound controls themselves.   This error message is generally an indication of a more serious error condition.

Could not refresh controls (Error 40504)

See Also

The **Refresh** method of the **RemoteData** control failed because of one of the following reasons:

- A connection could not be established.
- A result set could not be opened.
- A bound control failed to update.

Check the connection, the result set, and the control bindings (review information in Help about the bound control).   This error can also occur if the server or network unexpectedly drops the connection.   Generally, it indicates that the **rdoResultset** or connection is no longer usable.

See Also

Refresh Method

RemoteData Control

Invalid object (Error 40506)

See Also

An object other than an **rdoResultset** was assigned to the **Resultset** property.   Assign a valid **rdoResultset** object to the **Resultset** property.   The only object that can be assigned to the **Resultset** property of the **RemoteData** control is an **rdoResultset** object created with another **RemoteData** control or the **OpenResultset** method.

See Also

OpenResultset Method
rdoResultset Object, **rdoResultsets** Collection
Resultset Property

Invalid property value (Error 40505)

An inappropriate value has been assigned to a property.   Most properties only accept values of a certain type, and within a certain range.

To see the appropriate values for the property, search Help for the property in question.

Method cannot be called in RDC's current state (Error 40507)

See Also

A method has been called that cannot be completed while an **AddNew** or **Edit** operation is in progress.   For example, you cannot call the **Refresh** method while the **RemoteData control (RDC)** is editing an existing row or adding a new row.   Make sure **AddNew** and **Edit** operations are completed by executing the **Update** or **CancelUpdate** method, or by using one of the *Move* methods before calling the method that caused the error.

For additional information, search Help for the method in question.

See Also

AddNew Method
CancelUpdate Method
Edit Method
rdoResultset Object, **rdoResultsets** Collection
RemoteData Control
Update Method

One or more of the arguments is invalid (Error 40508)

See Also

A value of at least one of the arguments called by this method is beyond the valid range of values for the argument.   Use valid argument values to call the method.

For additional information, search Help for the method in question.

See Also

Remote Data Objects Method Summary

Out of memory (Error 40510)

More system resources were required than are available. This error might have one or more of the following causes and solutions:

- You have too many applications, documents, or source files open.
  Close any unnecessary applications, documents, or source files.
- You have terminate-and-stay-resident programs running.
  Eliminate terminate-and-stay-resident programs.
- You have too many device drivers loaded.
  Eliminate unnecessary device drivers.
- You have run out of space for **Public** variables.
  Reduce the number of **Public** variables.
- You have exhausted available TEMP or virtual memory space.
  Use the System Resource meter to view available system resources. Check available disk space.
- You have insufficient RAM to run the application or set of applications loaded in memory.
  Increase the amount of available RAM by installing additional memory, or reallocate memory to reduce the size of SmartDrive or other cache memory allocations.
- Your application has generated a memory leak; it allocates memory but does not free it when it is no longer needed.
- You have written reentrant code that is not properly executed or procedures that allocate excessive memory for arrays.

Property cannot be set in RDC's current state (Error 40513)

See Also

Some properties of the **RemoteData** control (RDC) cannot be set after you have programmatically set the **Resultset** property. If you create an **rdoResultset** object in code and set it to the **Resultset** property of the **RemoteData** control, the **RemoteData** control cannot automatically reset certain properties, such as **DataSourceName**, **Options**, **Password**, **QueryTimeout**, and **UserName**. The **RemoteData** control cannot reset these properties because the **rdoResultset** object was created outside the **RemoteData** control.

To reset these properties, close the current **rdoResultset** object, programmatically set the **RemoteData** control properties to new values, and call the **Refresh** method against the **RemoteData** control to rebuild the result set.

For additional information, search Help for the property in question.

See Also

- **DataSourceName** Property
- **Options** Property
- **Password** Property
- **QueryTimeout** Property
- **rdoResultset** Object, **rdoResultsets** Collection
- **RemoteData** Control
- **Resultset** Property
- **UserName** Property

Property not available in RDC's current state (Error 40514)

See Also

Because some properties cannot be accessed until a valid **RemoteData** control (RDC) / RDO connection is established, the state of the **RemoteData** control restricts read access for this property.   Set the appropriate **RemoteData** control properties and use the **Refresh** method to establish a connection.

For additional information, search Help for the property in question.

See Also

Refresh Method

RemoteData Control

Resultset is empty (Error 40509)

See Also

The result set is empty, so an operation that requires the **UpdateRow** method cannot be called for a nonexistent row.   Make sure the result set is not empty before calling **UpdateRow**.

See Also

UpdateRow Method

RemoteData Control

Resultset not available (Error 40511)

See Also

The **Resultset** property cannot find a valid result set because an error occurred while opening the result set, or the result set is closed.   Set the **SQL** property to a valid value and/or use the **Refresh** method against the **RemoteData** control.

See Also

Resultset Property
SQL Property
Refresh Method
RemoteData Control

The connection is not open (Error 40512)

See Also

Because there is no connection to a database, the **Connection** property cannot reference a valid database connection.   To establish a database connection, check the **SQL** property, set the **DataSourceName** or **Connection** property to a valid value, and use the **Refresh** method against the **RemoteData** control.

See Also

Connection Property
DataSourceName Property
Refresh Method
RemoteData Control
SQL Property

Type mismatch (Error 40515)

See Also

The wrong argument type was passed in the **RemoteData** control event parameter. Check the argument's values.

For additional information, search Help for the event in question.

See Also

Error Event

Reposition Event

QueryCompleted Event

Validate Event

Cannot connect to Remote Data Object (Error 40516)

See Also

The **RemoteData** control could not create a Remote Data Object.   This can occur if the Microsoft Remote Data Object 1.0 library was not registered correctly, or if the library is not present on the computer.

To manually register the Remote Data Object 1.0 library, type the following at the command prompt:

```
regsvr32 msrdo32.dll
```

If MSRDO32.DLL is not present on your system, reinstall Visual Basic and make sure you have purchased the Enterprise Edition.

See Also

RemoteData Control

Align Property (Remote Data)

Returns or sets a value that determines whether an object is displayed in any size anywhere on a form or whether it's displayed at the top, bottom, left, or right of the form and is automatically sized to fit the form's width.

Syntax

object.**Align** [**=** *number*]

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

Part	Description
object	An object expression that evaluates to a **RemoteData** control.
number	An integer specifying how an object is displayed, as described in Settings.

Settings

The settings for *number* are:

Constant	Value	Description
vbAlignNone	0	(Default in a non-MDI form) None−size and location can be set at design time or in code. This setting is ignored if the object is on an MDI form.
vbAlignTop	1	(Default in an MDI form) Top−object is at the top of the form, and its width is equal to the form's **ScaleWidth** property setting.
vbAlignBottom	2	Bottom−object is at the bottom of the form, and its width is equal to the form's **ScaleWidth** property setting.
vbAlignLeft	3	Left−object is at the left of the form, and its width is equal to the form's **ScaleWidth** property setting.
vbAlignRight	4	Right−object is at the right of the form, and its width is equal to the form's **ScaleWidth** property setting.

Remarks

These constants are listed in the Visual Basic (VB) object library of the Object Browser.

You can use the **Align** property to quickly create a toolbar or status bar at the top or bottom of a form. As a user changes the size of the form, an object with **Align** set to 1 or 2 automatically resizes to fit the width of the form.

BackColor, ForeColor Properties (Remote Data)

– **BackColor**
–returns or sets the background color of an object.
– **ForeColor**
–returns or sets the foreground color used to display text and graphics in an object.

Syntax

*object***.BackColor** [= *color*]

*object***.ForeColor** [= *color*]

The **BackColor** and **ForeColor** property syntaxes have these parts:

Part	Description
object	An object expression that evaluates to a **RemoteData** control.
color	A value or constant that determines the background or foreground colors of an object, as described in Settings.

Settings

Visual Basic uses the Microsoft Windows operating environment red-green-blue (RGB) color scheme. The settings for *color* are:

Setting	Description
Normal RGB colors	Colors specified by using the Color palette or by using the **RGB** or **QBColor** functions in code.
System default colors	Colors specified by system color constants listed in the Visual Basic (VB) object library in the Object Browser. The Windows operating environment substitutes the user's choices as specified in the Control Panel settings.

The default settings at design time are:

– **BackColor**
–set to the system default color specified by the constant **vbWindowBackground**.
– **ForeColor**
–set to the system default color specified by the constant **vbWindowText**.

Remarks

The valid range for a normal RGB color is 0 to 16,777,215 (&HFFFFFF). The high byte of a number in this range equals 0; the lower 3 bytes, from least to most significant byte, determine the amount of red, green, and blue, respectively. The red, green, and blue components are each represented by a number between 0 and 255 (&HFF). If the high byte isn't 0, Visual Basic uses the system colors, as defined in the user's Control Panel settings and by constants listed in the Visual Basic (VB) object library in the Object Browser.

To display text in the Windows operating environment, both the text and background colors must be solid. If the text or background colors you've selected aren't displayed, one of the selected colors may be dithered–that is, comprised of up to three different-colored pixels. If you choose a dithered color for either the text or background, the nearest solid color will be substituted.

Caption Property (Remote Data)

See Also

Returns or sets the text displayed in or next to a control.

Syntax

object.**Caption** [**=** *string*]

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

Part	Description
object	An object expression that evaluates to a **RemoteData** control. If *object* is omitted, the form associated with the active form module is assumed to be *object.*
string	A string expression that evaluates to the text displayed as the caption.

Remarks

When you create a new object, its default caption is the default **Name** property setting. This default caption includes the object name and an integer, such as Command1 or Form1. For a more descriptive label, set the **Caption** property.

You can use the **Caption** property to assign an access key to a control. In the caption, include an ampersand (&) immediately preceding the character you want to designate as an access key. The character is underlined. Press ALT plus the underlined character to move the focus to that control. To include an ampersand in a caption without creating an access key, include two ampersands (&&). A single ampersand is displayed in the caption and no characters are underlined.

See Also

Name Property

Drag Method (Remote Data)

See Also

Begins, ends, or cancels a drag operation of an object.

Syntax

*object***.Drag** *action*

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

Part	Description
object	An object expression that evaluates to a **RemoteData** control. If *object* is omitted, the object whose event procedure contains the **Drag** method is assumed.
action	Optional. A constant or value that specifies the action to perform, as described in Settings. If *action* is omitted, the default is to begin dragging the object.

Settings

The settings for *action* are:

Constant	Value	Description
vbCancel	0	Cancel drag operation.
vbBeginDrag	1	Begin dragging *object*.
vbEndDrag	2	End dragging and drop *object*.

Remarks

These constants are listed in the Visual Basic (VB) object library in the Object Browser.

Using the **Drag** method to control a drag-and-drop operation is required only when the **DragMode** property of the object is set to Manual (0). However, you can use **Drag** on an object whose **DragMode** property is set to Automatic (1 or **vbAutomatic**).

If you want the mouse pointer to change shape while the object is being dragged, use either the **DragIcon** or **MousePointer** property. The **MousePointer** property is only used if no **DragIcon** is specified.

See Also

DragIcon Property
DragMode Property
MousePointer Property

DragDrop Event (Remote Data)

See Also

Occurs when a drag-and-drop operation is completed as a result of dragging a control over a form or control and releasing the mouse button or using the **Drag** method with its *action* argument set to 2 (Drop).

Syntax

Private Sub *object*_**DragDrop(**[*index* **As Integer**,]*source* **As Control**, *x* **As Single**, *y* **As Single)**

The DragDrop event syntax has these parts:

Part	Description
object	An object expression that evaluates to a **RemoteData** control.
index	An integer that uniquely identifies a control if it's in a control array.
source	The control being dragged. You can include properties and methods with this argument, for example, `Source.Visible = 0`.
x, *y*	A number that specifies the current horizontal (*x*) and vertical (*y*) position of the mouse pointer within the target form or control. These coordinates are always expressed in terms of the target's coordinate system as set by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties.

Remarks

Use a DragDrop event procedure to control what happens after a drag operation is completed. For example, you can move the source control to a new location or copy a file from one location to another.

When multiple controls can potentially be used in a *source* argument:

– Use the **TypeOf** keyword with the **If** statement to determine the type of control used with *source.*

– Use the control's **Tag** property to identify a control, and then use a DragDrop event procedure.

Note Use the **DragMode** property and **Drag** method to specify the way dragging is initiated. Once dragging has been initiated, you can handle events that precede a DragDrop event with a DragOver event procedure.

See Also

Drag Method
DragIcon Property
DragMode Property
MouseDown, MouseUp Events
MouseMove Event

DragIcon Property (Remote Data)

See Also

Returns or sets the icon to be displayed as the pointer in a drag-and-drop operation.

Syntax

object.**DragIcon** [**=** *icon*]

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

Part	Description
object	An object expression that evaluates to a **RemoteData** control.
icon	Any code reference that returns a valid icon, such as a reference to a form's icon (`Form1.Icon`), a reference to another control's **DragIcon** property (`Text1.DragIcon`), or the **LoadPicture** function.

Settings

The settings for *icon* are:

Setting	Description
(none)	(Default) An arrow pointer inside a rectangle.
Icon	A custom mouse pointer. You specify the icon by setting it using the Properties window at design time. You can also use the **LoadPicture** function at run time. The file you load must have the .ICO filename extension and format.

Remarks

You can use the **DragIcon** property to provide visual feedback during a drag-and-drop operation−for example, to indicate that the source control is over an appropriate target. **DragIcon** takes effect when the user initiates a drag-and-drop operation. Typically, you set **DragIcon** as part of a MouseDown or DragOver event procedure.

Note At run time, the **DragIcon** property can be set to any object's **DragIcon** or **Icon** property, or you can assign it an icon returned by the **LoadPicture** function.

See Also

Drag Method
DragDrop Event
DragMode Property

DragMode Property (Remote Data)

See Also

Returns or sets a value that determines whether manual or automatic drag mode is used for a drag-and-drop operation.

Syntax

object.**DragMode** [**=** *number*]

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

Part	Description
object	An object expression that evaluates to a **RemoteData** control.
number	An integer specifying the drag mode, as described in Settings.

Settings

The settings for *number* are:

Constant	Value	Description
vbManual	0	(Default) Manual–requires using the **Drag** method to initiate a drag-and-drop operation on the source control.
vbAutomatic	1	Automatic–clicking the source control automatically initiates a drag-and-drop operation.

Remarks

These constants are listed in the Visual Basic (VB) object library of the Object Browser.

When **DragMode** is set to 1 (Automatic), the control doesn't respond as usual to mouse events.   Use the 0 (Manual) setting to determine when a drag-and-drop operation begins or ends; you can use this setting to initiate a drag-and-drop operation in response to a keyboard or menu command or to enable a source control to recognize a MouseDown event prior to a drag-and-drop operation.

Clicking while the mouse pointer is over a target object or form during a drag-and-drop operation generates a DragDrop event for the target object.   This ends the drag-and-drop operation.   A drag-and-drop operation may also generate a DragOver event.

Note   While a control is being dragged, it can't recognize other user-initiated mouse or keyboard events (KeyDown, KeyPress or KeyUp, MouseDown, MouseMove, or MouseUp). However, the control can receive events initiated by code or by a DDE link.

See Also

DragDrop Event
DragIcon Property
DragOver Event

DragOver Event (Remote Data)

See Also

Occurs when a drag-and-drop operation is in progress. You can use this event to monitor the mouse pointer as it enters, leaves, or rests directly over a valid target. The mouse pointer position determines the target object that receives this event.

Syntax

Private Sub *object*_**DragOver(**[*index* **As Integer**,]*source* **As Control**, *x* **As Single**, *y* **As Single**, *state* **As Integer)**

The DragOver event syntax has these parts:

Part	Description
object	An object expression that evaluates to a **RemoteData** control.
index	An integer that uniquely identifies a control if it's in a control array.
source	The control being dragged. You can refer to properties and methods with this argument, for example, `Source.Visible = False`.
x, *y*	A number that specifies the current horizontal (*x*) and vertical (*y*) position of the mouse pointer within the target form or control. These coordinates are always expressed in terms of the target's coordinate system as set by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties.
state	An integer that corresponds to the transition state of the control being dragged in relation to a target form or control: 0 = Enter (source control is being dragged within the range of a target). 1 = Leave (source control is being dragged out of the range of a target). 2 = Over (source control has moved from one position in the target to another).

Remarks

Use a DragOver event procedure to determine what happens after dragging is initiated and before a control drops onto a target. For example, you can verify a valid target range by highlighting the target (set the **BackColor** or **ForeColor** property from code) or by displaying a special drag pointer (set the **DragIcon** property from code).

Use the *state* argument to determine actions at key transition points. For example, you might highlight a possible target when *state* is set to 0 (Enter) and restore the object's previous appearance when *state* is set to 1 (Leave).

When an object receives a DragOver event when *state* is set to 0 (Enter):

- If the source control is dropped on the object, that object receives a DragDrop event.
- If the source control isn't dropped on the object, that object receives another DragOver event when *state* is set to 1 (Leave).

Note Use the **DragMode** property and **Drag** method to specify the way dragging is initiated. For suggested techniques with the *source* argument, see Remarks for the DragDrop event topic.

See Also

Drag Method
DragDrop Event
DragIcon Property
DragMode Property
MouseDown, MouseUp Events
MouseMove Event

Enabled Property (Remote Data)

See Also

Returns or sets a value that determines whether a form or control can respond to user-generated events.

Syntax

object.**Enabled** [**=** *boolean*]

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

Part	Description
object	An object expression that evaluates to a **RemoteData** control. If *object* is omitted, the form associated with the active form module is assumed to be *object.*
boolean	A Boolean expression specifying whether *object* can respond to user-generated events.

Settings

The settings for *boolean* are:

Setting	Description
True	(Default) Allows *object* to respond to events.
False	Prevents *object* from responding to events.

Remarks

The **Enabled** property allows forms and controls to be enabled or disabled at run time. For example, you can disable objects that don't apply to the current state of the application. You can also disable a control used purely for display purposes, such as a text box that provides read-only information.

See Also

Visible Property

Font Property (Remote Data)

Returns a **Font** object.

Syntax

*object***.Font**

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

Remarks

Use the **Font** property of an object to identify a specific **Font** object whose properties you want to use.   For example, the following code changes the **Bold** property setting of a **Font** object identified by the **Font** property of a **RemoteData** object:

```
MSRDC1.Font.Bold = True
```

Height, Width Properties (Remote Data)

See Also

Return or set the dimensions of an object.

Syntax

object.**Height** [= *number*]

object.**Width** [= *number*]

The **Height** and **Width** property syntaxes have these parts:

Part	Description
object	An object expression that evaluates to a **RemoteData** control.
number	A numeric expression specifying the dimensions of an object, as described in Settings.

Measurements are calculated from the center of the control's border so that controls with different border widths align correctly.   These properties use the scale units of a control's container.

Remarks

The values for these properties change as the object is sized by the user or by your code. Maximum limits of these properties for all objects are system-dependent.

If you set the **Height** and **Width** properties for a printer driver that doesn't allow these properties to be set, no error occurs and the size of the paper remains as it was.   If you set **Height** and **Width** for a printer driver that allows only certain values to be specified, no error occurs and the property is set to whatever the driver allows.   For example, you could set **Height** to 150 and the driver would set it to 144.

Use the **Height**, **Width**, **Left**, and **Top** properties for operations or calculations based on an object's total area, such as sizing or moving the object.   Use the **ScaleLeft**, **ScaleTop**, **ScaleHeight**, and **ScaleWidth** properties for operations or calculations based on an object's internal area, such as drawing or moving objects within another object.

See Also

Left, **Top** Properties

Move Method

HelpContextID Property (Remote Data)

See Also

Returns or sets an associated context number for an object. Used to provide context-sensitive Help for your application.

Syntax

object.**HelpContextID** [**=** *number*]

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

Part	Description
object	An object expression that to a **RemoteData** control. If *object* is omitted, the form associated with the active form module is assumed to be *object.*
number	A numeric expression specifying the context number of the Help topic associated with *object*.

Settings

The settings for *number* are:

Setting	Description
0	(Default) No context number specified.
> 0	An integer specifying a valid context number.

Remarks

For context-sensitive Help on an object in your application, you must assign the same context number to both *object* and to the associated Help topic when you compile your Help file.

If you've created a Microsoft Windows operating environment Help file for your application and set the application's **HelpFile** property, when a user presses the F1 key, Visual Basic automatically calls Help and searches for the topic identified by the current context number.

The current context number is the value of **HelpContextID** for the object that has the focus. If **HelpContextID** is set to 0, then Visual Basic looks in the **HelpContextID** of the object's container, and then that object's container, and so on. If a nonzero current context number can't be found, the F1 key is ignored.

Note Building a Help file requires the Microsoft Windows Help Compiler, which is included with the Visual Basic, Professional Edition.

See Also

HelpContext, **HelpFile** Properties

Index Property (Remote Data)

See Also

Returns or sets the number that uniquely identifies a control in a control array. Available only if the control is part of a control array.

Syntax

object[**(***number***)**]**.Index**

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

Part	Description
object	An object expression that evaluates to a **RemoteData** control.
number	A numeric expression that evaluates to an integer that identifies an individual control within a control array.

Settings

The settings for *number* are:

Setting	Description
No value	(Default) Not part of a control array.
0 to 32,767	Part of an array. Specifies an integer greater than or equal to 0 that identifies a control within a control array. All controls in a control array have the same **Name** property. Visual Basic automatically assigns the next integer available within the control array.

Remarks

Because control array elements share the same **Name** property setting, you must use the **Index** property in code to specify a particular control in the array. **Index** must appear as an integer (or a numeric expression evaluating to an integer) in parentheses next to the control array name, for example, `MyButtons(3)`. You can also use the **Tag** property setting to distinguish one control from another within a control array.

When a control in the array recognizes that an event has occurred, Visual Basic calls the control array's event procedure and passes the applicable **Index** setting as an additional argument. This property is also used when you create controls dynamically at run time with the **Load** statement or remove them with the **Unload** statement.

Although Visual Basic assigns, by default, the next integer available as the value of **Index** for a new control in a control array, you can override this assigned value and skip integers. You can also set **Index** to an integer other than 0 for the first control in the array. If you reference an **Index** value in code that doesn't identify one of the controls in a control array, a Visual Basic run-time error occurs.

Note To remove a control from a control array, change the control's **Name** property setting, and delete the control's **Index** property setting.

See Also

Name Property

Tag Property

Left, Top Properties (Remote Data)

See Also

– **Left**

–returns or sets the distance between the internal left edge of an object and the left edge of its container.

– **Top**

–returns or sets the distance between the internal top edge of an object and the top edge of its container.

Syntax

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

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

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

Part	Description
object	An object expression that to a **RemoteData** control.
value	A numeric expression specifying distance.

Remarks

The **Left** and **Top** properties are measured in units whose size depends on the coordinate system of its container.   The values for these properties change as the object is moved by the user or by code.

For both properties, you can specify a single-precision number.

Use the **Left**, **Top**, **Height**, and **Width** properties for operations based on an object's external dimensions, such as moving or resizing.

See Also

Move Method

MouseDown, MouseUp Events (Remote Data)

See Also

Occur when the user presses (MouseDown) or releases (MouseUp) a mouse button.

Syntax

Private Sub *object*_**MouseDown(**[*index* **As Integer**,]*button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single)**

Private Sub *object* _**MouseUp(**[*index* **As Integer**,]*button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single)**

The MouseDown and MouseUp event syntaxes have these parts:

Part	Description
object	Returns an object expression that evaluates to a **RemoteData** control.
index	Returns an integer that uniquely identifies a control if it's in a control array.
button	Returns an integer that identifies the button that was pressed (MouseDown) or released (MouseUp) to cause the event. The *button* argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Only one of the bits is set, indicating the button that caused the event.
shift	Returns an integer that corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the *button* argument is pressed or released. A bit is set if the key is down. The *shift* argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The *shift* argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT were pressed, the value of *shift* would be 6.
x, *y*	Returns a number that specifies the current location of the mouse pointer. The *x* and *y* values are always expressed in terms of the coordinate system set by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties of the object.

Remarks

Use a MouseDown or MouseUp event procedure to specify actions that will occur when a given mouse button is pressed or released. Unlike the Click and DblClick events, MouseDown and MouseUp events enable you to distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

The following applies to both Click and DblClick events:

– If a mouse button is pressed while the pointer is over a form or control, that object "captures" the mouse and receives all mouse events up to and including the last MouseUp event. This implies that the *x*, *y* mouse-pointer coordinates returned by a mouse event may not always be in the internal area of the object that receives them.

– If mouse buttons are pressed in succession, the object that captures the mouse after the first press receives all mouse events until all buttons are released.

If you need to test for the *button* or *shift* arguments, you can use constants listed in the Visual Basic (VB) object library in the Object Browser to define the bits within the argument:

Constant	Value	Description
vbLeftButton	1	Left button is pressed.
vbRightButton	2	Right button is pressed.
vbMiddleButton	4	Middle button is pressed.
vbShiftMask	1	SHIFT key is pressed.

vbCtrlMask	2	CTRL key is pressed.
vbAltMask	4	ALT key is pressed.

The constants then act as bit masks you can use to test for any combination of buttons without having to figure out the unique bit field value for each combination.

Note You can use a MouseMove event procedure to respond to an event caused by moving the mouse. The *button* argument for MouseDown and MouseUp differs from the *button* argument used for MouseMove. For MouseDown and MouseUp, the *button* argument indicates exactly one button per event; for MouseMove, it indicates the current state of all buttons.

See Also

MouseMove Event

MouseMove Event (Remote Data)

See Also

Occurs when the user moves the mouse.

Syntax

Private Sub *object*_**MouseMove(**[*index* **As Integer**,] *button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single)**

The MouseMove event syntax has these parts:

Part	Description
object	An object expression that evaluates to a **RemoteData** control.
index	An integer that uniquely identifies a control if it's in a control array.
button	An integer that corresponds to the state of the mouse buttons in which a bit is set if the button is down. The *button* argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. It indicates the complete state of the mouse buttons; some, all, or none of these three bits can be set, indicating that some, all, or none of the buttons are pressed.
shift	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys. A bit is set if the key is down. The *shift* argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The *shift* argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT were pressed, the value of *shift* would be 6.
x, *y*	A number that specifies the current location of the mouse pointer. The *x* and *y* values are always expressed in terms of the coordinate system set by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties of the object.

Remarks

The MouseMove event is generated continually as the mouse pointer moves across objects. Unless another object has captured the mouse, an object recognizes a MouseMove event whenever the mouse position is within its borders.

If you need to test for the *button* or *shift* arguments, you can use constants listed in the Visual Basic (VB) object library in the Object Browser to define the bits within the argument:

Constant	Value	Description
vbLeftButton	1	Left button is pressed.
vbRightButton	2	Right button is pressed.
vbMiddleButton	4	Middle button is pressed.
vbShiftMask	1	SHIFT key is pressed.
vbCtrlMask	2	CTRL key is pressed.
vbAltMask	4	ALT key is pressed.

The constants then act as bit masks you can use to test for any combination of buttons without having to figure out the unique bit field value for each combination.

You test for a condition by first assigning each result to a temporary integer variable and then comparing the *button* or *shift* arguments to a bit mask. Use the **And** operator with each argument to test if the condition is greater than zero, indicating the key or button is pressed, for example:

```
LeftDown = (Button And vbLeftButton) > 0
CtrlDown = (Shift And vbCtrlMask) > 0
```

Then, in a procedure, you can test for any combination of conditions, for example:

```
If LeftDown And CtrlDown Then
```

Note You can use MouseDown and MouseUp event procedures to respond to events caused by pressing and releasing mouse buttons.

The *button* argument for MouseMove differs from the *button* argument for MouseDown and MouseUp. For MouseMove, the *button* argument indicates the current state of all buttons; a single MouseMove event can indicate that some, all, or no buttons are pressed. For MouseDown and MouseUp, the *button* argument indicates exactly one button per event.

Any time you move a window inside a MouseMove event, it can cause a cascading event. MouseMove events are generated when the window moves underneath the pointer. A MouseMove event can be generated even if the mouse is perfectly stationary.

See Also

MouseDown, MouseUp Events

Parent Property (Remote Data)

Returns the form on which a control is located.

Syntax

object.**Parent**

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

Remarks

Use the **Parent** property to access the properties, methods, or controls of a control's parent form, for example:

```
MSRDC1.Parent.MousePointer = 4
```

The **Parent** property is useful in an application in which you pass controls as arguments. For example, you could pass a control variable to a general procedure in a module, and use the **Parent** property to access its parent form.

ShowWhatsThis Method (Remote Data)

See Also

Displays a selected topic in a Help file using the What's This popup provided by Windows 95 Help.

Important This method requires the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating systems.

Syntax

object.**ShowWhatsThis**

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

Remarks

The **ShowWhatsThis** method is very useful for providing context-sensitive Help from a context menu in your application. The method displays the topic identified by the **WhatsThisHelpID** property of the object specified in the syntax.

See Also

WhatsThisHelpID Property

Tag Property (Remote Data)

See Also

Returns or sets an expression that stores any extra data needed for your program. Unlike other properties, the value of the **Tag** property isn't used by Visual Basic; you can use this property to identify objects.

Syntax

object.**Tag** [= *expression*]

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

Part	Description
object	An object expression that evaluates to a **RemoteData** control.
expression	A string expression identifying the object. The default is a zero-length string ("").

Remarks

You can use this property to assign an identification string to an object without affecting any of its other property settings or causing side effects. The **Tag** property is useful when you need to check the identity of a control that is passed as a variable to a procedure.

See Also

Name Property

Visible Property (Remote Data)

Returns or sets a value indicating whether an object is visible or hidden.

Syntax

object.**Visible** [= *boolean*]

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

Part	Description
object	An object expression that evaluates to a **RemoteData** control.
boolean	A Boolean expression specifying whether the object is visible or hidden.

Settings

The settings for *boolean* are:

Setting	Description
True	(Default) Object is visible.
False	Object is hidden.

Remarks

To hide an object at startup, set the **Visible** property to **False** at design time.   Setting this property in code enables you to hide and later redisplay a control at run time in response to a particular event.

WhatsThisHelpID Property (Remote Data)

See Also

Returns or sets an associated context number for an object. Use to provide context-sensitive Help for your application using the What's This pop-up in Windows 95 Help.

Important This property requires the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating systems.

Syntax

object.**WhatsThisHelpID** [= *number*]

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

Part	Description
object	An object expression that evaluates to a **RemoteData** control.
number	A numeric expression specifying a Help context number, as described in Settings.

Settings

The settings for *number* are:

Setting	Description
0	(Default) No context number specified.
>0	An integer specifying the valid context number for the What's This topic associated with the object.

Remarks

Visual Basic applications can support either of two different models for context-sensitive Help.

- Window 3.x
- Windows 95

The Windows 3.x model uses the F1 key to start Windows Help and load the topic identified by the **HelpContextID** property. The Windows 95 model typically uses the What's This button in the upper right corner of the window to start Windows Help and load a topic identified by the **WhatsThisHelpID** property. Use the **WhatsThisHelp** property to select between the two context-sensitive models.

See Also

HelpContextID Property
ShowWhatsThis Method

ZOrder Method (Remote Data)

Places a specified **MDIForm**, **Form**, or control at the front or back of the z-order within its graphical level. Doesn't support named arguments.

Syntax

object.**ZOrder** *position*

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

Part	Description
object	Optional. An object expression that evaluates to a **RemoteData** control. If *object* is omitted, the form with the focus is assumed to be *object*.
position	Optional. Integer indicating the position of *object* relative to other instances of the same *object*. If position is 0 or omitted, *object* is positioned at the front of the z-order. If position is 1, *object* is positioned at the back of the z-order.

Remarks

The z-order of objects can be set at design time by choosing the Bring To Front or Send To Back menu command from the Edit menu.

Container Property (Remote Data)

Returns or sets the container of a control on a Form.   Not available at design time.

Syntax

Set *object*.**Container** [= *container*]

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

Part	Description
object	An object expression that evaluates to a **RemoteData** control.
container	An object expression that evaluates to an object that can serve as a container for other controls, as described in Remarks.

Remarks

The following controls can contain other controls:

- **Frame** control
- **PictureBox** control.

Object Property (Remote Data)

Returns a reference to a property or method of a custom control which has the same name as a property or method automatically extended to the control by Visual Basic.

Syntax

object.**Object**[.*property*|.*method*] [= *value*]

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

Part	Description
object	An object expression that evaluates to a **RemoteData** control.
property	Property of the custom control that is identical to the name of a Visual Basic-supplied property.
method	Method of the custom control that is identical to the name of a Visual Basic-supplied method.
value	Constant or value of the property or method.

Remarks

Visual Basic supplies some or all of a standard set of properties and methods to custom controls in a Visual Basic project. It is possible for a custom control to define a property or method which has the same name as one of these standard properties or methods. When this occurs, Visual Basic automatically uses the property or method it supplies instead of the one with the same name defined in the custom control. The **Object** property allows you to bypass the Visual Basic-supplied property or method and use the identically named property or method defined in the custom control.

For example, the **Tag** property is a property supplied to all custom controls in a Visual Basic project. If a custom control in a project has the name `ctlDemo`, and you access the **Tag** property using this syntax:

```
ctlDemo.Tag
```

Visual Basic automatically uses the **Tag** property it supplies. However, if the custom control defines its own **Tag** property and you want to access that property, you use the **Object** property in this syntax:

```
ctlDemo.Object.Tag
```

Visual Basic automatically extends some or all of the following properties and methods to custom controls in a Visual Basic project:

Properties

Align	**DragIcon**	**LinkMode**	**TabIndex**
Cancel	**DragMode**	**LinkItem**	**TabStop**
Container	**Enabled**	**LinkTimeout**	**Tag**
DataChanged	**Height**	**LinkTopic**	**Top**
DataField	**HelpContextID**	**Name**	**Visible**
DataSource	**Index**	**Negotiate**	**WhatsThisHelpID**
Default	**Left**	**Parent**	**Width**

Methods

Drag	**LinkRequest**	**SetFocus**
LinkExecute	**LinkSend**	**ShowWhatsThis**
LinkPoke	**Move**	**ZOrder**

If you use a property or method of a custom control and don't get the behavior you expect, check to see if the property or method has the same name as one of those shown in the preceding list. If the names match, check the documentation provided with the custom control to see if the behavior matches the Visual Basic-supplied property or method.   If the behaviors aren't identical, you may need to use the **Object** property to access the feature of the custom control that you want.

Appearance Property (Remote Data)

See Also

Returns or sets the paint style of controls on an **MDIForm** or **Form** object at run time. Read-only at run time.

Syntax

[*object*]**.Appearance**

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

Settings

The **Appearance** property settings are:

Setting	Description
0	Flat. Paints controls and forms with without visual effects.
1	(Default) 3D. Paints controls with three-dimensional effects.

Remarks

If set to 1 at design time, the **Appearance** property draws controls with three-dimensional effects.

See Also

BackColor, **ForeColor** Properties

MousePointer Property (Remote Data)

See Also

Returns or sets a value indicating the type of mouse pointer displayed when the mouse is over a particular part of an object at run time.

Syntax

object.**MousePointer** [**=** *value*]

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

Part	Description
object	An object expression that evaluates to a **RemoteData** control.
value	An integer specifying the type of mouse pointer displayed, as described in Settings.

Settings

The settings for *value* are:

Setting	Description
0	(Default) Shape determined by the object.
1	Arrow.
2	Cross (cross-hair pointer).
3	I-Beam.
4	Icon (small square within a square).
5	Size (four-pointed arrow pointing north, south, east, and west).
6	Size NE SW (double arrow pointing northeast and southwest).
7	Size N S (double arrow pointing north and south).
8	Size NW SE (double arrow pointing northwest and southeast).
9	Size W E (double arrow pointing west and east).
10	Up Arrow.
11	Hourglass (wait).
12	No Drop.
13	Arrow and hourglass.
14	Arrow and question mark.
15	Size all (customizable under Microsoft Windows NT 3.51)
99	Custom icon specified by the **MouseIcon** property.

Remarks

You can use this property when you want to indicate changes in functionality as the mouse pointer passes over controls on a form or dialog box. The Hourglass setting (11) is useful for indicating that the user should wait for a process or operation to finish.

Note If your application doesn't call the **DoEvents** function and isn't a 32-bit application, it overrides all **MousePointer** settings for all controls and other applications. If your application calls DoEvents, the **MousePointer** property may temporarily change when over a custom control.

See Also

DragIcon Property
MouseIcon Property
MouseMove Event

MouseIcon Property (Remote Data)

See Also

Sets a custom mouse icon.

Syntax

*object***.MouseIcon = LoadPicture(***pathname***)**

*object***.MouseIcon =** *picture*

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

Part	Description
object	An object expression that evaluates to a **RemoteData** control.
pathname	A string expression specifying the path and filename of the file containing the custom icon.
picture	The **Picture** property of a **Form** object, **PictureBox** control, or **Image** control.

Remarks

The **MouseIcon** property provides a custom icon that is used when the **MousePointer** property is set to 99.

Although Visual Basic does not create cursor (.CUR) files, you can use the **MouseIcon** property to load either cursor or icon files.   This provides your program with easy access to custom cursors of any size, with any desired hot spot location.   The 32-bit version of Visual Basic does not load animated cursor (.ANI) files, even though 32-bit versions of Windows support these cursors.

See Also

DragIcon Property

MousePointer Property

Remote Data Constants

See Also

You can use the Object Browser to browse the list of built-in constants. From the View menu, choose Object Browser, select the appropriate library, and then select the Constants object. Scroll the list in the Methods/Properties box to see the complete list of constants.

RDO provides built-in constants that you can use with methods or properties. These constants all begin with the letters "rd" and are documented with the method or property to which they apply.

Attributes Property Constants

Constant	Value	Description
rdFixedColumn	1	The column size is fixed (default for numeric columns).
rdVariableColumn	2	The column size is variable (text columns only).
rdAutoIncrColumn	16	The column value for new rows is automatically incremented to a unique **Long** integer that can't be changed.
rdUpdatableColumn	32	The column value can be changed.

BOFAction Property Constants

Constant	Value	Description
rdMoveFirst	0	**MoveFirst** (Default): Keeps the first row as the current row.
rdBOF	1	**BOF**: Moving past the beginning of an **rdoResultset** triggers the **RemoteData** control's Validate event on the first row, followed by a Reposition event on the invalid (**BOF**) row. At this point, the Move Previous button on the **RemoteData** control is disabled.

CursorDriver, rdoDefaultCursorDriver Property Constants

Constant	Value	Description
rdUseIfNeeded	0	The ODBC driver will choose to use the appropriate style of cursors. It will use server-side cursors if they are available.
rdUseOdbc	1	The RDO layer will use the ODBC cursor library. This gives better performance for small result sets but degrades quickly for larger result sets.
rdUseServer	2	Use server-side cursors. For most large operations this will give better performance, but can cause more network traffic.

Direction Property Constants

Constant	Value	Description
rdParamInput	0	(Default) The parameter is used to pass information to the procedure.
rdParamInputOutput	1	The parameter is used to pass information both to and from the procedure.
rdParamOutput	2	The parameter is used to return information from the procedure as in an output parameter in SQL.
rdParamReturnValue	3	The parameter is used to pass the return value from a procedure.

EditMode Property Constants

Constant	Value	Description

rdEditNone	0	No editing operation is in progress.
rdEditInProgress	1	The **Edit** method has been invoked, and the current row is in the copy buffer.
rdEditAdd	2	The **AddNew** method has been invoked, and the current row in the copy buffer is a new row that hasn't been saved in the database.

EOFAction Property Constants

Constant	Value	Description
rdMoveLast	0	**MoveLast** (Default): Keeps the last row as the current row.
rdEOF	1	**EOF**: Moving past the end of an **rdoResultset** triggers the **RemoteData** control's Validation event on the last row, followed by a Reposition event on the invalid (**EOF**) row. At this point, the Move Next button on the **RemoteData** control is disabled.
rdAddNew	2	**AddNew**: Moving past the last row triggers the **RemoteData** control's Validation event to occur on the current row, followed by an automatic **AddNew**, followed by a Reposition event on the new row.

Error Event *CancelDisplay* Argument Constants

Constant	Value	Description
rdDataErrContinue	0	Continue.
rdDataErrDisplay	1	(Default) Display the error message.

LockType Property, OpenResultset Method *locktype* Argument Constants

Constant	Value	Description
rdConcurReadOnly	1	(Default) Cursor is read-only. No updates are allowed.
rdConcurLock	2	Pessimistic concurrency. Cursor uses the lowest level of locking sufficient to ensure the row can be updated.
rdConcurRowVer	3	Optimistic concurrency based on row ID. Cursor compares row ID in old and new rows to determine if changes have been made since the row was last accessed.
rdConcurValues	4	Optimistic concurrency based on row values. Cursor compares data values in old and new rows to determine if changes have been made since the row was last accessed.

Options Property, Execute and OpenResultset Methods *options* Argument, Constants

Constant	Value	Description
rdAsyncEnable	32	Execute operation asynchronously.

Prompt Property, OpenConnection Method *prompt* Argument Constants

Constant	Value	Description
rdDriverPrompt	0	The driver manager displays the ODBC (Open Database Connectivity) Data Sources dialog box. The connection string used to establish the connection is constructed from the data source name (DSN) selected and completed by the user via the dialog boxes. Or, if no DSN is chosen and the **DataSourceName** property is empty, the default DSN is used.
rdDriverNoPrompt	1	The driver manager uses the connection string provided in *connect*. If sufficient information is not

		provided, the **OpenConnection** method returns a trappable error.
rdDriverComplete	2	If the connection string provided includes the DSN keyword, the driver manager uses the string as provided in *connect*, otherwise it behaves as it does when **rdDriverPrompt** is specified.
rdDriverCompleteRequired	3	(Default) Behaves like **rdDriverComplete** except the driver disables the controls for any information not required to complete the connection.

rdoLocaleID Property Constants

Constant	Value	Description
rdLocaleSystem	0	System
rdLocaleEnglish	1	English
rdLocaleFrench	2	French
rdLocaleGerman	3	German
rdLocaleItalian	4	Italian
rdLocaleJapanese	5	Japanese
rdLocaleSpanish	6	Spanish
rdLocaleChinese	7	Chinese

ResultsetType and Type Properties, OpenResultset Method *type* Argument Constants

Constant	Value	Description
rdOpenForwardOnly	0	Opens a forward-only–type **rdoResultset** object. (Default)
rdOpenKeyset	1	Opens a keyset-type **rdoResultset** object.
rdOpenDynamic	2	Opens a dynamic-type **rdoResultset** object.
rdOpenStatic	3	Opens a static-type **rdoResultset** object.

SQLRetCode Property Constants

Constant	Description
rdSQLSuccess	The operation is successful.
rdSQLSuccessWithInfo	The operation is successful, and additional information is available.
rdSQLNoDataFound	No additional data is available.
rdSQLError	An error occurred performing the operation.
rdSQLInvalidHandle	The handle supplied is invalid.

Type Property (rdoColumn, rdoParameter) Constants

Constant	Value	Description
rdTypeCHAR	1	Fixed-length character string. Length set by **Size** property.
rdTypeNUMERIC	2	Signed, exact, numeric value with precision p and scale s (1 p 15; 0 s p).
rdTypeDECIMAL	3	Signed, exact, numeric value with precision p and scale s (1 p 15; 0 s p).
rdTypeINTEGER	4	Signed, exact numeric value with precision 10, scale 0 (signed: -231 n 231-1; unsigned: 0 n 232-1).
rdTypeSMALLINT	5	Signed, exact numeric value with precision 5, scale 0 (signed: -32,768 n 32,767, unsigned: 0 n 65,535).
rdTypeFLOAT	6	Signed, approximate numeric value with mantissa

		precision 15 (zero or absolute value 10-308 to 10308).
rdTypeREAL	7	Signed, approximate numeric value with mantissa precision 7 (zero or absolute value 10-38 to 1038).
rdTypeDOUBLE	8	Signed, approximate numeric value with mantissa precision 15 (zero or absolute value 10-308 to 10308).
rdTypeDATE	9	Date – data source dependent.
rdTypeTIME	10	Time – data source dependent.
rdTypeTIMESTAMP	11	TimeStamp – data source dependent.
rdTypeVARCHAR	12	Variable-length character string. Maximum length 255.
rdTypeLONGVARCHAR	-1	Variable-length character string. Maximum length determined by data source.
rdTypeBINARY	-2	Fixed-length binary data. Maximum length 255.
rdTypeVARBINARY	-3	Variable-length binary data. Maximum length 255.
rdTypeLONGVARBINARY	-4	Variable-length binary data. Maximum data source dependent.
rdTypeBIGINT	-5	Signed, exact numeric value with precision 19 (signed) or 20 (unsigned), scale 0; (signed: -263 n 263-1; unsigned: 0 n 264-1).
rdTypeTINYINT	-6	Signed, exact numeric value with precision 3, scale 0; (signed: -128 n 127, unsigned: 0 n 255).
rdTypeBIT	-7	Single binary digit.

Type Property (rdoPreparedStatement) Constants

Constant	Value	Description
rdQSelect	0	Select query
rdQAction	1	Action query
rdQProcedure	2	Procedural query

Validate Event *action* Argument Constants

Constant	Value	Description
rdDataActionCancel	0	Cancel the operation when the **Sub** exits.
rdDataActionMoveFirst	1	**MoveFirst** method.
rdDataActionMovePrevious	2	**MovePrevious** method.
rdDataActionMoveNext	3	**MoveNext** method.
rdDataActionMoveLast	4	**MoveLast** method.
rdDataActionAddNew	5	**AddNew** method.
rdDataActionUpdate	6	**Update** operation (not **UpdateRow**).
rdDataActionDelete	7	**Delete** method.
rdDataActionBookmark	8	The **Bookmark** property has been set.
rdDataActionClose	9	The **Close** method.
rdDataActionUnload	10	The form is being unloaded.

See Also

RemoteData Control

Remote Data Objects and Collections

Documents the SetupWizard application.   For information
about the Setup Toolkit, see the Visual Basic Help file.

Documents Visual Basic for Windows.

Documents the Data Access application.

Documents the Data Manager application.

Tutorials for learning to use Visual Basic for Windows.

Documents Microsoft Support Services.

Lists the applications written in Visual Basic that demonstrate
techniques discussed in the printed documentation.

Documents the custom controls provided with the Professional Edition.

Documents the Crystal Reports application.

Documents the segmented hypergraphic editor for creating hotspots within graphics for use in authoring Help files.

Documents the installation tools for ODBC.

Documents the ODBC driver for SQL Server databases.

Documents the VisData sample application.

Documents Windows functions as used in the C programming language.

Documents the Code Profiler add-in.

Documents Remote Automation, the Component Manager, Remote Data Objects (RDO), and the RemoteData control provided with the Enterprise Edition.

Documents the SourceSafe add-in for administrators.

Documents the SourceSafe add-in for users.

Text Files

Microsoft Visual Basic 4.0 includes additional information in the following files:

Text File	Description
APILOD.TXT	Describes how to use the API Text Viewer.
LABELS.TXT	Contains information about mailing labels.
PACKING.LST	Lists all files on the distribution disks provided with Visual Basic.
VB4DLL.TXT	Contains additional information about developing dynamic link libraries (DLLs) to use with Visual Basic.
VB4Q&A.WRI	Contains frequently asked questions concerning Visual Basic version 4.0 and the appropriate answers.
WIN31API.TXT	Contains procedure, constant, and type declarations for 16-bit versions of Windows API functions.
WIN32API.TXT	Contains symbolic constants for 32-bit versions of Windows API functions.
WINMMSYS.TXT	Contains procedure, constant, and type declarations for Windows 3.1 multimedia API functions.

The Automation Manager could not be started. 'msgtext'

The Automation Manager could not be started on this computer for the reason given in the system error message appended at 'msgtext.'

For example, an `Out of Memory` error would prevent the Automation Manager from starting.

The Automation Manager was started with the following network protocols: 'protocols'.

Startup message (Information). The Automation Manager attempts to start using all the RPC network server protocols currently installed on the computer. A list of the protocols that were used successfully appears at 'protocols.'

You can view the list of RPC server protocols on a computer by accessing the following Windows Registry key:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Rpc\ServerProtocols

The Automation Manager was unable to use network protocol 'protocol'.   'msgtext'

Startup message (Warning).   The Automation Manager attempts to start with all the RPC network server protocols currently installed on the computer.   The Automation Manager was unable to use the protocol 'protocol' for the reason given in the system error message appended at 'msgtext.'

You can view the list of RPC server protocols on a computer by accessing the following Windows Registry key:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Rpc\ServerProtocols

The command line parameter 'parameter' was not recognized.

Startup message (Warning). The Automation Manager was started with an invalid command-line parameter, which was ignored. Valid parameters are:

Parameter	Description
/REGSERVER	Registers the Automation Manager in the Windows Registry.
/UNREGSERVER	Unregisters the Automation Manager.
/AUTOMATION	Accepted but ignored.
/EMBEDDING	Accepted but ignored.
/HIDDEN	Starts the Automation Manager without any visible window, so that it runs invisibly. This is particularly useful when the Automation Manager runs on a Win32 workstation. If the Automation Manager is started with this parameter there is no way for the user to close it.

The file 'file' could not be loaded.   'msgtext'

This error occurs during Setup.   The Automation Manager has a dependency on the file named in the message, and that file cannot be found on the computer where Setup is being run, for the reason appended at 'msgtext'.

For example, Automation Manager depends on AUTPRX32.DLL.

The network protocol 'protocol' was needed but was not available. 'msgtext'

The Automation Manager was unable to use the indicated network protocol, for the reason appended at 'msgtext.'

This error can occur when a client computer passes a reference to an object provided by an OLE server on the client computer, or on another network computer.   (The client computer must itself be capable of running the Automation Manager for the first scenario to occur.)

The Automation Manager attempts to connect to the object using the same protocol that was used to pass the reference.   If this protocol is not available on the network computer where the Automation Manager is running, the error occurs.

For example, the Named Pipes protocol (ncacn_np) is supported as an RPC client protocol under Windows 95, but not as an RPC server protocol.   You can view the lists of RPC client and server protocols on a computer by accessing the following Windows Registry keys:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Rpc\ClientProtocols

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Rpc\ServerProtocols

On the client computer, the call that attempted to pass the object reference fails with the error &H800706D0, RPC_S_PROTSEQ_NOT_FOUND.

The preference setting 'setting' had an invalid value. The default setting will be used instead.

Automation Manager's RemoteActivationPolicy setting was invalid.   The default setting, CreateIfKey (2), will be used.   Valid values are:

Setting	Description
0	CreateNone. Do not allow creation of any CLSID.
1	CreateAll. Allow creation of any CLSID. Not recommended.
2	CreateIfKey. Allow creation of only those CLSIDs that include the subkey AllowRemoteActivation=Y
3	CreateIfAcl. Allow creation of a CLSID only if the user making the request has KEY_QUERY_VALUE permission on the CLSID key.

The location of the RemoteActivationPolicy preference is:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Automation Manager\RemoteActivationPolicy

There was a timeout processing a call to this machine.   'msgtext'

A call to a method of an object on this machine has timed out for the reason appended at 'msgtext.'

The timeout occurs if the client computer waits more than a specified number of milliseconds for an object, because another client computer is making a call to the same object.

This error only occurs when the Automation Manager serializes requests from two client computers that are accessing the same object.   It does not occur when two client computers have separate objects of the same type, nor when OLE serializes requests to an out-of-process OLE server that is single-threaded.

The number of milliseconds for the timeout is specified in the Automation Manager's CallTimeout preference setting:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Automation Manager\CallTimeout

The default value is an unsigned four-byte integer, 0xFFFFFFFF, which represents over a hundred years, and thus is effectively infinite.   Changing this preference affects all calls to methods of objects on this computer; hence changing it is not recommended.

There was an error accessing preferences in the registry. 'msgtext'

Automation Manager was unable to access its preferences in the Windows Registry for the reason appended at 'msgtext.'

For example, an error will occur if the Automation Manager is executed from a login that does not have read permission to the subkeys in the Windows Registry that contain Automation Manager's preferences:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Automation Manager\CallTimeout

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Automation Manager\RemoteActivationPolicy

There was an error adding information to the registry.   'msgtext'

Setup for the Automation Manager was unable to add the necessary entries to the Windows Registry for the reason appended at 'msgtext.'   For example, an error will occur if Setup is executed from a login that does not have write access to the Windows Registry.

There was an error connecting to an object of type 'type'.   'msgtext'

A client computer has a reference to an object on this computer, and has attempted to pass that reference to a third computer.   The third computer was unable to connect to the object for the reason appended at 'msgtext.'

'Type' is the fully qualified class name (programmatic ID), for example `Customer.Order`.

There was an error creating an object of type 'type' for 'user'. 'msgtext'

A client computer requested creation of an object of the specified type, for example by using the **CreateObject** function, or the **Set** statement with the **New** operator.   The request failed for the reason given in the appended 'msgtext.'

If the class the object belongs to is in the Windows Registry of the network computer the Automation Manager is running on, 'type' is the fully qualified class name (programmatic ID), for example `Customer.Order`.

If the object type is not registered, for example when the OLE server that supplies it has not been installed on the network computer, 'type' is the CLSID of the object, as it appears in the Windows Registry of the client computer.

The user name associated with the request to create the object is inserted at 'user.'

This form of the error occurs only if both of the following are true:

- The Automation Manager must be running under the Microsoft Windows NT operating system
- The client computer must be using an Authentication level other than None (1).

Otherwise, the Automation Manager cannot determine the user name, and the form of the error without the user information is used.

There was an error creating an object of type 'type'.   'msgtext'

A client computer requested creation of an object of the specified type, for example by using the **CreateObject** function, or the **Set** statement with the **New** operator.   The request failed for the reason given in the appended 'msgtext.'

If the class the object belongs to is in the Windows Registry of the network computer the Automation Manager is running on, 'type' is the fully qualified class name (programmatic ID).

For example, suppose the Customer server is installed on the network computer, the Automation Manager is using CreateIfKey security, and the CLSID entry for the Order class does not have the subkey AllowRemoteActivation=Y.   The error message refers to the object type `Customer.Order`.

If the object type is not registered, for example when the OLE server that supplies it has not been installed on the network computer, 'type' is the CLSID of the object, as it appears in the Windows Registry of the client computer.

There was an error processing a call to this machine. 'msgtext'

A call to a method of an object provided by an OLE server running on this computer has failed for the reason appended at 'msgtext.'

This error occurs when the Automation Manager has received the method call from the client computer, but is unable to make the cross-process call to the OLE server, or made the cross-process call but failed to receive the result.

This error occurs only for cross-process communication failures. It does not occur for errors raised in an OLE server, or for OLE exceptions; such errors are simply returned to the client application.

There was an error removing information from the registry. 'msgtext'

The Automation Manager was run with the **/UNREGSERVER** parameter, and was unable to remove the necessary entries from the Windows Registry for the reason appended at 'msgtext.'   For example, an error will occur if the Automation Manager is executed in this fashion from a login that does not have write access to the Windows Registry.

Using the Component Manager

Overview of the Microsoft Component Manager

How-to Information
A listing of task-oriented topics.

Interface Information
A functional listing of menu commands and dialog boxes.

How-to Information

Interface Information

Component Manager Dialog Boxes
Component Manager Menus

Component Manager Dialog Boxes

About Command (Help Menu)

Displays the About dialog box for the Component Manager.

Add Catalog Command (Catalog Menu)

See Also

Displays the Add Component Catalog dialog box, in which you can add additional component catalogs to the Scope pane.

See Also

Add Component Catalog Dialog Box
Create New Catalog Dialog Box
Customizing a Component Catalog

Add OLE Components Command (Results Menu)

See Also

Displays the Add OLE Components dialog box, in which you can add OLE components from type libraries, the system registry, .EXE and .DLL files, or other component catalogs.

See Also

Add OLE Components Dialog Box
Adding Components to a Component Catalog
Details Tab, Properties Dialog Box
Install Dialog Box

Clear Search Criteria Command (Search Menu)

See Also

Clears all search selections in the Criteria pane for the selected component catalog, and returns the date criteria to start 1/1/70 and end on the current date (today's date).   This will not clear the search criteria for other component catalogs in the Component Manager.

Toolbar shortcut:

See Also

Property Tags Tab, Properties Dialog Box
Finding and Sorting Components
Tagging Components
Component Manager Toolbar

Close Result Pane Outline Command (View Menu)

See Also

Closes the bottom level of open folders in the Results pane.   Successive clicks will close higher levels until all folders are closed.

Toolbar shortcut:

See Also

Component Manager Toolbar

Contents Command (Help Menu)

Displays this Help file.

Copy Command (Results Menu)

See Also

Copies programming information about a service onto the Clipboard, where it can then be pasted into code modules. Enabled only if an item is selected in the Results pane.

– If the item is an .EXE folder, the title of the .EXE and its detailed description are copied onto the Clipboard.
– If the item is a class folder, a **Dim** statement for the class is copied onto the Clipboard.
– If the item is a method or property, a sample statement using the method with all associated parameter placeholders is copied onto the Clipboard.

See Also

Delete Command (Results Menu)

Delete Command (Results Menu)

See Also

Removes all component catalog references associated with the selected item.   If the item is an .EXE file or a class, all services related to that item will be deleted from the catalog.

Note   This command does not delete the catalog from the file system.

See Also

Copy Command (Results Menu)

Exit Command (File Menu)

Exits the Component Manager.

Export Component Catalog Command (Catalog Menu)

See Also

Displays the Export Component Catalog dialog box, in which you can export the contents of one component catalog to another.

See Also

Export Component Catalog Dialog Box

Install Command (Results Menu)

See Also

Displays the Install dialog box, in which you can install a component on your local system for evaluation or register it for remote use on a network server.

See Also

Details Tab, Properties Dialog Box
Add OLE Components Dialog Box
Overview of the Microsoft Component Manager
Install Dialog Box

Layout Command (View Menu)

See Also

Displays four additional submenus that you use to determine which panes are displayed in the Component Manager.

See Also

[All Panes (View Menu, Layout Submenu)]
[Component Manager Toolbar]
[Criteria & Result Pane (View Menu, Layout Submenu)]
[Result Pane Only (View Menu, Layout Submenu)]
[Scope & Result Pane (View Menu, Layout Submenu)]

Result Pane Only Command (View Menu, Layout Submenu)

See Also

Displays only the Results pane.

Toolbar shortcut: ▭

See Also

Component Manager Toolbar

Scope & Result Pane Command (View Menu, Layout Submenu)

See Also

Displays the Scope and Results panes.

Toolbar shortcut: ▭

See Also

Component Manager Toolbar

Criteria & Result Pane Command (View Menu, Layout Submenu)

See Also

Displays the Criteria and Results panes.
Toolbar shortcut: ▭

See Also

Component Manager Toolbar

All Panes Command (View Menu, Layout Submenu)

See Also

Displays the Criteria, Results, and Scope panes.
Toolbar shortcut:

See Also

Component Manager Toolbar

Open Result Pane Outline Command (View Menu)

See Also

Opens the top level of folders in the Results pane. Successive clicks will open lower levels until all folders are open.

Toolbar shortcut:

See Also

Component Manager Toolbar

Properties Command (Catalog Menu)

See Also

Displays the Catalog Properties dialog box for the currently selected component catalog. You can use this dialog box to add, edit, and remove a catalog's properties and values, if you have write permission to the component catalog.

See Also

[Catalog Properties Dialog Box]

[Customizing a Component Catalog]

Properties Command (Results Menu)

See Also

Displays the Properties dialog box for the currently selected component.   You can edit information in the Description, Property Tags, Details, and Associated Files tabs, if you have write permission to the component catalog.

See Also

Adding Information About Components
Associated Files Tab, Properties Dialog Box
Description Tab, Properties Dialog Box
Property Tags Tab, Properties Dialog Box
Details Tab, Properties Dialog Box

Remove Catalog Command (Catalog Menu)

See Also

Removes the reference to the currently selected component catalog.

See Also

Add OLE Components Dialog Box
Adding Components to a Component Catalog

Find Results Now Command (Search Menu)

See Also

Combines the current criteria into a search query that is applied against the current component catalog.   The results of the query are displayed in hierarchical order in the Results pane in the currently defined sort order.

Toolbar shortcut:

See Also

Component Manager Toolbar
Finding and Sorting Components
Property Tags Tab, Properties Dialog Box
Tagging Components

Show Collection Lists Command (View Menu)

See Also

Displays the user-defined named collections in the Criteria pane.

Toolbar shortcut:

See Also

Component Manager Toolbar
Edit Named Collections Dialog Box

Show Filter & Sort Criteria Command (View Menu)

See Also

Displays the currently selected component catalog's filter and sort criteria.
Toolbar shortcut:

See Also

Component Manager Toolbar
Finding and Sorting Components
Property Tags Tab, Properties Dialog Box
Tagging Components

Switch to Local/Remote Catalog Command (Catalog Menu)

See Also

Toggles the selected component catalog from local to remote.

Note This command is enabled only if you have previously exported the component catalog to a remote location with the Export Component Catalog dialog box.

See Also

Switching Between Local and Remote Execution

Export Component Catalog Dialog Box

User Options Command (View Menu)

See Also

Displays the User Options dialog box, in which you can set the menu bar, status bar, and window position options.

Toolbar shortcut:

See Also

User Options Dialog Box

Component Manager Toolbar

Component Manager Menus

File Menu
View Menu
Catalog Menu
Search Menu
Results Menu
Help Menu

Component Manager Toolbar

_

Catalog Menu

Add Catalog
Remove Catalog
Export Component Catalog
Switch to Local/Remote Catalog
Properties

File Menu

Exit

_

Help Menu

Contents
About

_

Results Menu

Properties
Install
Add OLE Components
Copy
Delete

_

Search Menu

Clear Search Criteria
Find Results Now

_

View Menu

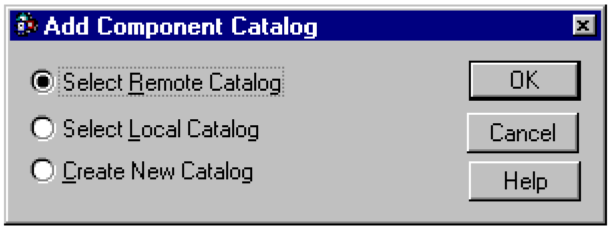
- Layout
    - Result Pane Only
    - Scope & Result Pane
    - Criteria & Result Pane
    - All Panes
- Show Filter & Sort Criteria
- Show Collection Lists
- Open Result Pane Outline
- Close Result Pane Outline
- User Options

About Dialog Box

Displays a dialog box with information about the Component Manager.

Add Component Catalog Dialog Box

See Also



Use the Add Component Catalog dialog box to add additional component catalogs to the Scope pane.

- To open the Add Component Catalog dialog box, click the Add Catalog command on the Catalog menu, or click the right mouse button over the Scope pane, and then click Add Catalog.
- To close the Add Component Catalog dialog box, click Cancel or double-click the Control-menu box. To accept your selections, click OK.

Dialog Box Options

Select Remote Catalog

Selecting this option allows you to browse available ODBC (Open Database Connectivity) data sources for previously created remote component catalogs. If you make a valid selection, a local .MDB file is created at the default .MDB directory path of your system, and tables are attached to the remote component catalog. The catalog is then added to the Scope pane.

Note Data is not automatically imported to your local component catalog when you add a remote component catalog. To do this, you must use the Export Component Catalog command to copy data from the remote ODBC data source to your local component catalog.

Select Local Catalog

Selecting this option allows you to add a local component catalog to the Scope pane. It displays the Component Catalog Locator dialog box, which allows you to select the path to the component catalog you want to add. If the selected catalog is not valid, the operation is canceled. If the selected catalog is valid, it is added to the Scope pane.

Create New Catalog

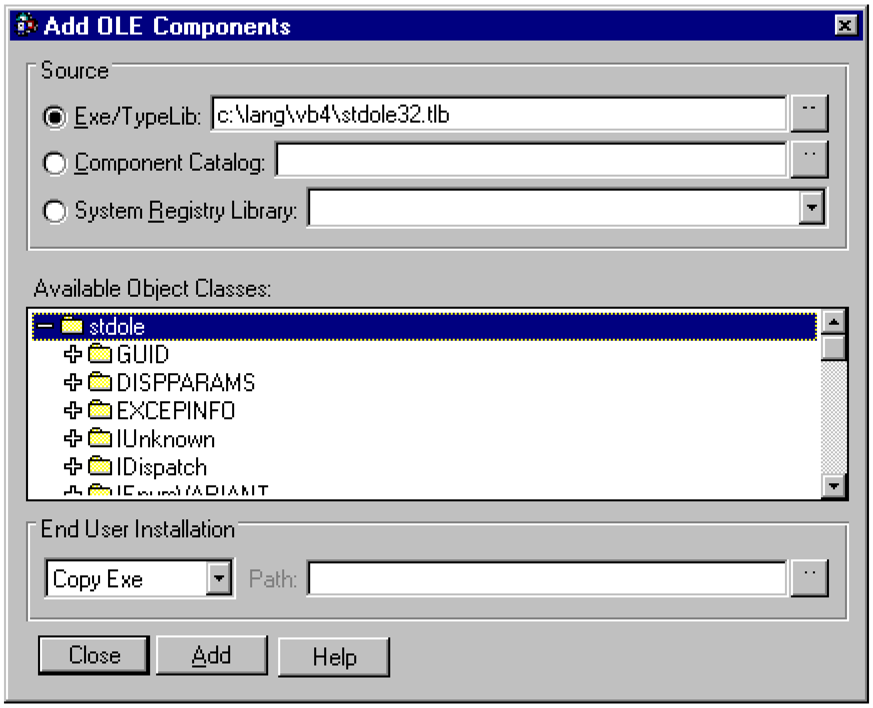
Selecting this option displays the Create New Catalog dialog box, which allows you to create a new component catalog. You can define its title and filename, as well as the native language used (such as English, French, and so forth).

See Also

Add Catalog Command (Catalog Menu)
Adding Components to a Component Catalog
Create New Catalog Dialog Box
Export Component Catalog Dialog Box

Add OLE Components Dialog Box

See Also



Exe/TypeLib: c:\lang\vb4\stdole32.tlb

Available Object Classes:

```
− stdole
   + GUID
   + DISPPARAMS
   + EXCEPINFO
   + IUnknown
   + IDispatch
```

End User Installation: Copy Exe

Use the Add OLE Components dialog box to add OLE server references from .EXE files, type libraries, the system registry, or other component catalogs to the active component catalog, as well as define installation parameters for users who want to use the available OLE servers.

Note The Add OLE Components dialog box does not add the actual OLE server files. It adds only descriptions of the OLE servers and installation information to the active component catalog.

− To open the Add OLE Components dialog box, click Add OLE Components on the Results menu, or click the right mouse button over the Results pane, and then click Add OLE Components

− To close the Add OLE Components dialog box, click Close or double-click the Control-menu box.

Dialog Box Options

Source

You must select one of the following three sources of OLE components:

− Exe/TypeLib

−Allows you to select executable (.EXE, .DLL) or TypeLib (.TLB, .OLB) files to add to your component catalog. Clicking the Browse button displays a dialog box in which you specify a path to the OLE component you want to add to the component catalog.

Not all OLE components are self-contained .EXE files. Some store data in an external TypeLib file. If you want to add such a component, select the .TLB or .OLB file in the File Browse dialog box. The related .EXE file will automatically be loaded along with the TypeLib file.

Note Not all .EXE files are valid OLE servers. As such, you will get an error if you attempt to add an .EXE file that is not an OLE server to the active component catalog.

– Component Catalog
–Allows you to open a local or remote component catalog (.MDB file) from which you can select OLE component descriptions to add to the active component catalog.
– System Registry Library
–Allows you to select an OLE component listed in your system registry. Once you select an item, it is displayed with the other available class modules and their members in the Available Object Classes box.

Note Adding very large OLE servers (such as Microsoft Excel, which contains over 2,300 members) can take a long time to complete. In such cases, it is more time effective to add only those classes and members that you will need.

Available Object Classes

Displays the currently available classes and members, based on the selected component source.

End User Installation

Users of your component catalog can view information on OLE servers, or even install them for use in their projects. The End User Installation options allow you to determine if and how each component will be installed if users attempt to install it locally on their machines. These options do not affect remote component installations.

Once you select a class in the Available Object Classes list, you can select an install option and path which users are prompted for when they attempt to install that object. If the source of the component is another component catalog, the Type and Path options are disabled, because the installation information is taken from the source component catalog.

Type

There are three installation types:

– None
–Use this option when you don't want other users to be able to install that particular OLE server
–you just want to provide information about it. If you select this option, when users attempt to install the OLE server, they will get an error message alerting them that they cannot install it.
– Copy Server
–Installs a copy of the selected OLE server to the user's local machine and registers it.
– Use Setup App
–Performs the OLE server installation by running a setup application program that you provide. This is useful, for example, in cases where the OLE server requires additional files to operate correctly. The setup program performs the installation of the files, putting them in the correct locations on the system.

Path

Used only when the Use Setup App option is selected. This is the path to the setup application program.

Close

Closes the Add OLE Components dialog box.

Add

Adds the selected OLE component to the current component catalog.

See Also

Adding Components to a Component Catalog
Details Tab, Properties Dialog Box
Install Dialog Box
Overview of the Microsoft Component Manager

Adding Components to a Component Catalog

See Also

One way to add components to a new or existing component catalog is from an .EXE or TypeLib (.TLB, .OLB) file.

To add components to a new or existing component catalog

1. Select the catalog in the Scope pane to which you want to add the component(s).
2. Click the right mouse button anywhere over the Results pane, and click the Add OLE Components command on the context menu.
3. From the Add OLE Components dialog box, select the Exe/TypeLib option, and then click the Browse button to the right side of it.
4. Select an OLE server .EXE file, and then click Open.   (You could also select a TypeLib file.)
5. Select the component just added to the Available Object Classes list box, and then change the End User Installation option, if necessary.
6. Click the Add button, and then close the Add OLE Components dialog box.

The new component should now be visible in the Results pane.   Repeat this process to add references to other servers.

Note   When you add a new component, the registry (.VBR) file and the TypeLib (.TLB, .OLB) file (if needed) for the component are automatically added to the list in the Associated Files tab.

You can also add components from other sources, such as other component catalogs or the system registry.   The installation procedure for these other sources is very similar to the one previously outlined.

See Also

Add OLE Components Dialog Box

Associated Files Tab, Properties Dialog Box

Customizing a Component Catalog

Adding Information About Components

See Also

Once you add a component to a component catalog, you can then add detailed information about it. You can enter this information in the Description tab of the component properties dialog box. It's important that you take the time to add this information, because it lets other developers know how to use the component, what it does, who wrote it, any usage constraints that might be associated with it, and so forth.

To add information about a component

1. Open the component by selecting it in the Results pane. Click the **+** outline icons to expose the various classes it contains, in addition to the members contained in each class.
2. Select an item in the hierarchy.
3. Click the right mouse button, and then click Properties.
4. Click the Description tab and add the information.

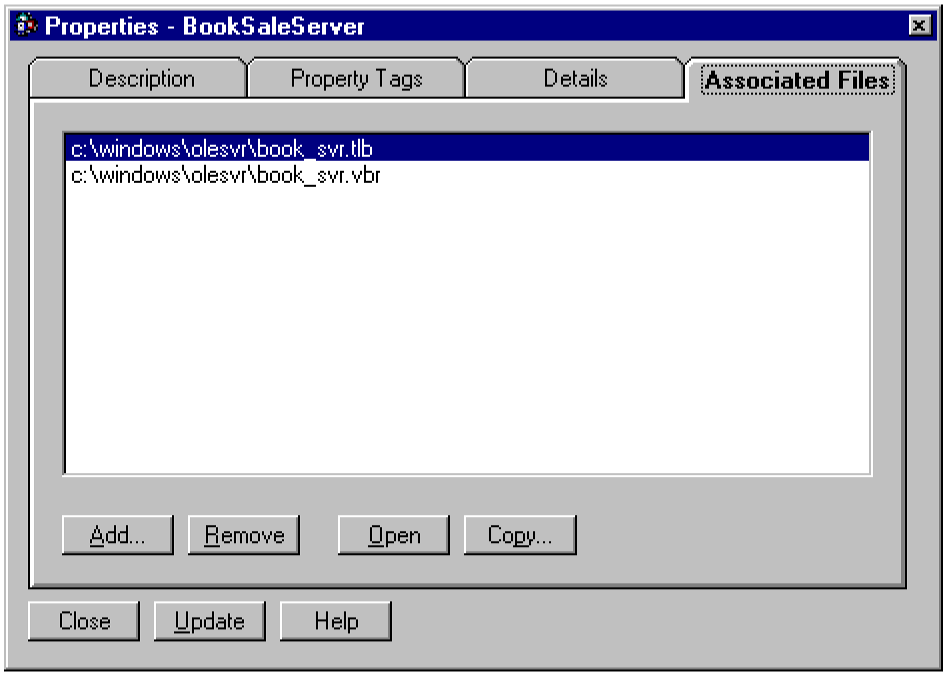
Using the Property Tags tab, you can also select global information relating to all classes in a component or all members in a class by setting the properties on the component and class respectively. When you change the tag for an item, the tags for all items subordinate to it are also changed.

See Also

Add OLE Components Dialog Box
Associated Files Tab, Properties Dialog Box
Description Tab, Properties Dialog Box
Property Tags Tab, Properties Dialog Box
Tagging Components

Associated Files Tab, Properties Dialog Box

See Also



c:\windows\olesvr\book_svr.tlb
c:\windows\olesvr\book_svr.vbr

Use the Associated Files tab to associate support files to be included with a particular OLE component, such as specification files, instructions, Help files, sample client source code files, and so forth. Any type of file can be associated with the selected component.

– To open the Associated Files tab, select a component in the Results pane, and then click the right mouse button. Click the Properties command, and then click the Associated Files tab.

– To close the Associated Files tab, click Close or double-click the Control-menu box.

Tab Options

Add

Displays a dialog box in which you can select a file to associate with the selected component. To add a file to the associated file list, select a file, and then click Open.

Note When you add a new component, the registry (.VBR) file and the TypeLib (.TLB/.OLB) file (if needed) for the component are automatically added to the list in the Associated Files tab.

Remove

Removes the selected file from the associated file list. You can remove only one file at a time.

Open

Opens the currently selected associated file. For example, if you have associated a .TXT file and you click Open, the .TXT file is opened in Notepad.

Note If you have not previously associated the file type in Windows with a particular program, the Open command will not be able to open the file. You can prevent this from happening by associating each file you might possibly open with a program.

Copy

Copies the selected associated file to another location. To copy a file, select a file in the associated file list, click Copy, specify a path and filename, and then click OK.

Close
: Closes the Properties dialog box.

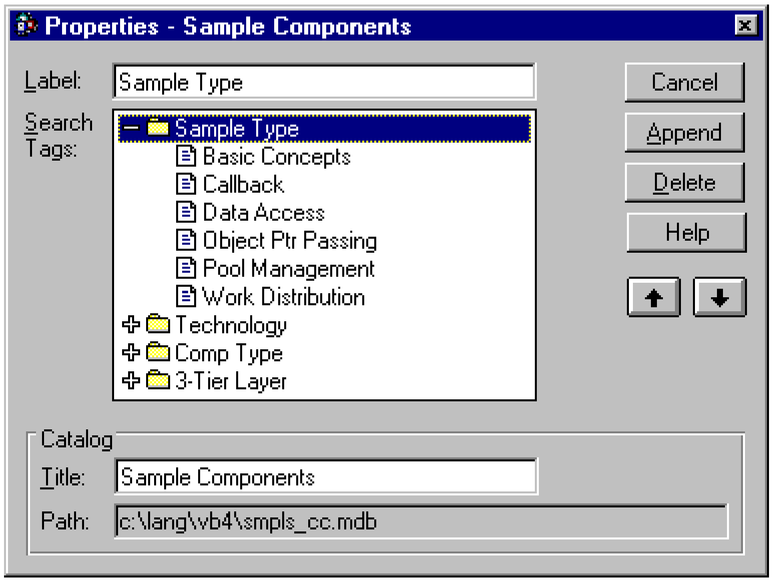
Update
: Updates the component with the currently defined values.

See Also

Add OLE Components Dialog Box
Description Tab, Properties Dialog Box
Details Tab, Properties Dialog Box
Property Tags Tab, Properties Dialog Box

Catalog Properties Dialog Box

See Also



Label: Sample Type

Search Tags:
- Sample Type
  - Basic Concepts
  - Callback
  - Data Access
  - Object Ptr Passing
  - Pool Management
  - Work Distribution
- Technology
- Comp Type
- 3-Tier Layer

Title: Sample Components

Path: c:\lang\vb4\smpls_cc.mdb

Use the Catalog Properties dialog box to alter existing property tags and values, or enter new ones. This dialog box lists the current property and value criteria available in the Criteria pane.

– To open the Catalog Properties dialog box, select a component catalog in the Scope pane, click the right mouse button, and then click the Properties command.

– To close the Catalog Properties dialog box, click Close or double-click the Control-menu box.

Dialog Box Options

Label

The name of the currently selected property or value. You can enter new names for properties and values here. The default value for Label is New Value for values, and New Property for properties.

Search Tags

The hierarchical listing of properties and their values.

Title

The name of the component catalog with which the properties and values are associated.

Path

The path of the component catalog. This is for information purposes and is read-only.

Close

Closes the Properties dialog box.

Append

Adds a new property or value to the Search Tags list.

To add a new property, select an existing property and click Append. A property with the default name of New Property containing a value with the default name of New Value is created. To add a new value to a property, click an existing value of the property for which you want to add the new value, and then click Append. A value with the default name of New Value is created.

Delete

Removes the currently selected property or value from the Search Tags list, and removes all component associations with that value.

See Also

Customizing a Component Catalog
Finding and Sorting Components
Overview of the Microsoft Component Manager
Property Tags Tab, Properties Dialog Box

Component Manager Toolbar

See Also

The toolbar provides easy access to the Component Manager's command set.

The Show Result Pane Only button displays only the Results pane.

The Show Scope & Result Panes button displays the Scope and Results panes only.

The Show Criteria & Result Panes button displays the Criteria pane and Results panes only.

The Show All Panes button displays the Criteria, Scope, and Results panes.

The Show Filter & Sort Criteria button displays a component catalog's filter and sort criteria in the Criteria pane, as defined by the component catalog's administrator.

The Show Collection Lists button displays user-defined named collections in the Criteria pane.

The Clear Search Criteria button clears all search criteria selections in the currently selected component catalog.   It does not affect the search criteria selections of other component catalogs.   When clicked, it sets the starting date criteria to 1/1/70 and the end date criteria to the current date.

The Find Results Now button combines the current criteria into a query which is applied against the current component catalog.   The results of the query are displayed in the Results pane.

The Expand Outline button opens the top level of folders displayed in the Results pane (OLE components) to display the classes contained in them.   Another click displays the members contained in each class.

The Close Outline button closes the bottom-most level of folders displayed in the Results pane (OLE components), hiding the members contained in them.   Another click hides the classes contained in each OLE component.

The Set User Options button displays the User Options dialog box, which allows you to display or not display the menu and status bars, as well as save the main window location.

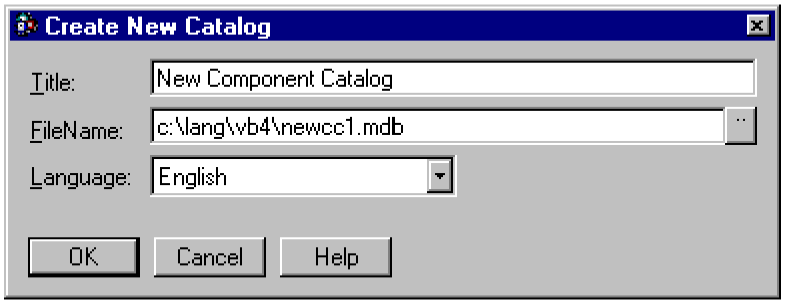
The Help button displays the Help file.

See Also

Overview of the Microsoft Component Manager
User Options Dialog Box

Create New Catalog Dialog Box

See Also



Title: New Component Catalog
FileName: c:\lang\vb4\newcc1.mdb
Language: English

Use the Create New Catalog dialog box to define a title and filename for a new component catalog.

– To open the Create New Catalog dialog box, click the right mouse button over the Scope pane, click Add Catalog, select the Create New Catalog option, and click OK.

– To close the Create New Catalog dialog box, click Close or double-click the Control-menu box.

Dialog Box Options

Title

The name of the component catalog displayed in the Component Manager Scope pane.

Filename

The name of the .MDB file that comprises the component catalog. You can change the path by entering it manually, or by using the Browse button to specify a new filename.

The default name for new component catalogs is "NEWCCxxx.MDB", where xxx ranges from 1–999 based on whether files with these names already exist at the specified path. New component catalogs have no initial properties or items. You cannot overwrite existing component catalogs.

Language

The base language used in the component catalog. Examples of base languages include English, French, and Japanese.

See Also

Add Component Catalog Dialog Box
Customizing a Component Catalog
Overview of the Microsoft Component Manager

Customizing a Component Catalog

See Also

To customize a component catalog

1. Select a catalog in the Scope pane.   Note that you must have write permissions to the file and location of the component catalog you want to alter.
2. Click the right mouse button over the Scope pane and select Properties.

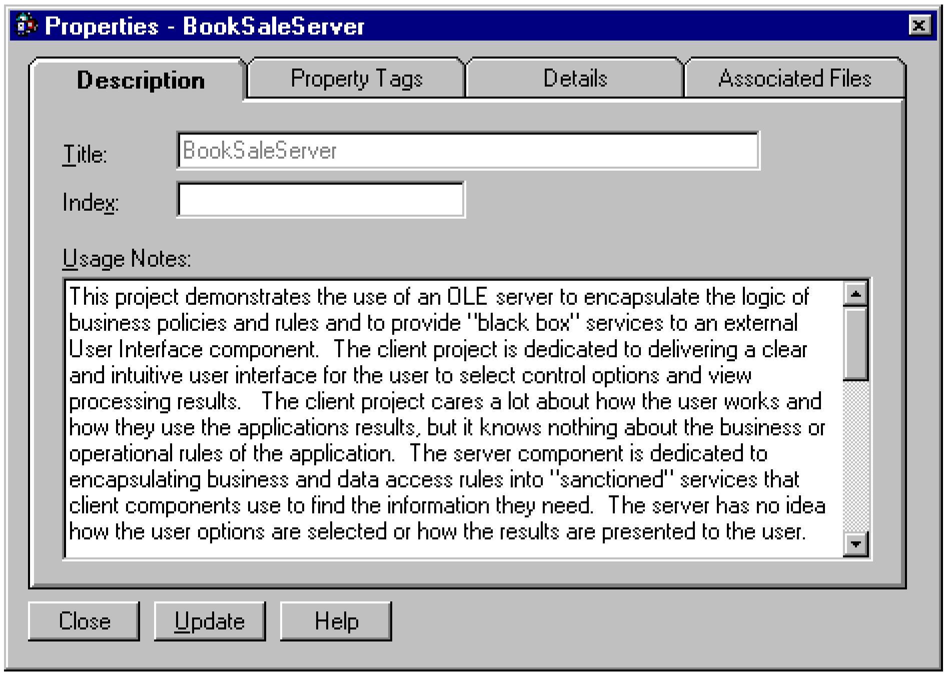
   This displays the Properties dialog box, in which you can define properties and values used to tag catalog components.   For example, you could create a property label called "Component Type" and add values such as "Finance," "Marketing," "Order Processing," "Sales," and "Tax." Another property might be "Author," containing as values a list of individuals or development groups that built the components in the catalog.

See Also

Catalog Properties Dialog Box
Finding and Sorting Components
Overview of the Microsoft Component Manager
Property Tags Tab, Properties Dialog Box
Tagging Components

Description Tab, Properties Dialog Box

See Also



Title: BookSaleServer

Index:

Usage Notes:

This project demonstrates the use of an OLE server to encapsulate the logic of business policies and rules and to provide "black box" services to an external User Interface component. The client project is dedicated to delivering a clear and intuitive user interface for the user to select control options and view processing results. The client project cares a lot about how the user works and how they use the applications results, but it knows nothing about the business or operational rules of the application. The server component is dedicated to encapsulating business and data access rules into "sanctioned" services that client components use to find the information they need. The server has no idea how the user options are selected or how the results are presented to the user.

Use the Description tab to view or change (if you have write permission) the description of the currently selected item in the Results pane.

When you make changes to a component's description, the Properties dialog box immediately updates to reflect the changes.

– To open the Description tab, select a component in the Results pane, and then click the right mouse button. Click the Properties command, and then click the Description tab.
– To close the Description tab, click Close or double-click the Control-menu box.

Tab Options

Title
The name of the selected property or value. This field is read-only.

Index
The Help index number of the selected property or value used for invoking context-sensitive Help when you double-click the selected item in the Results pane. This field is read-only.

Usage Notes
Contains any text an administrator wants to share with users, such as the purpose of the item, its usage constraints, its author, and so forth.

Close
Closes the Properties dialog box.

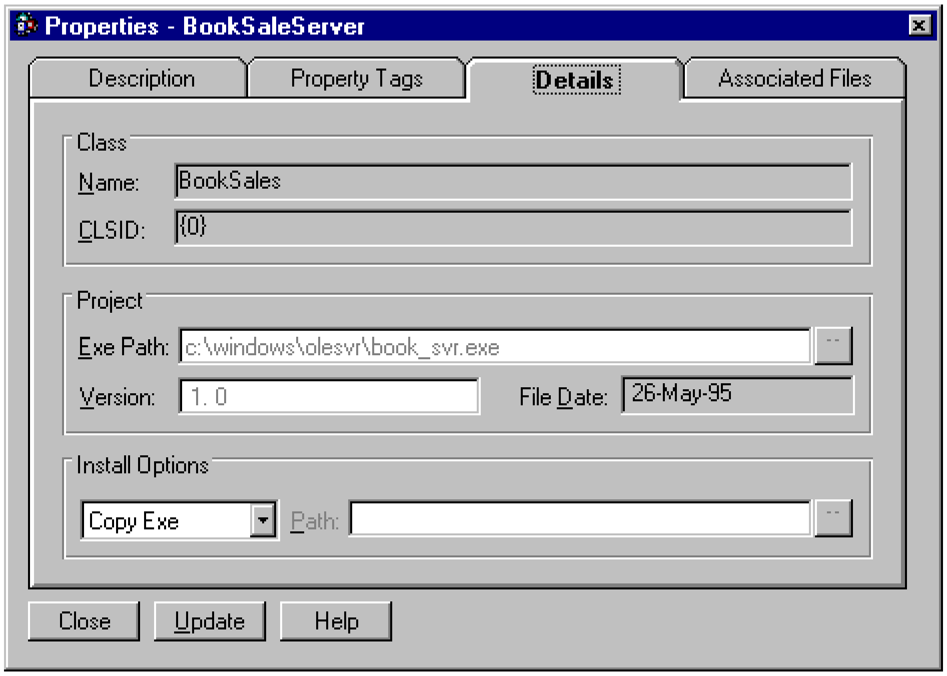
Update
Updates the component with the currently defined values.

See Also

Associated Files Tab, Properties Dialog Box
Customizing a Component Catalog
Details Tab, Properties Dialog Box
Property Tags Tab, Properties Dialog Box

Details Tab, Properties Dialog Box

See Also



Class

Name: BookSales

CLSID: {0}

Project

Exe Path: c:\windows\olesvr\book_svr.exe

Version: 1. 0

File Date: 26-May-95

Install Options

Copy Exe

Path:

Use the Details tab to provide a description and technical details about the selected component.

– To open the Details tab, select a component in the Results pane, and then click the right mouse button. Click the Properties command, and then click the Details tab.

– To close the Details tab, click Close or double-click the Control-menu box.

Tab Options

Class

– Name

–The class name of the selected item (read-only).

– CLSID

–The class ID number of the selected item (read-only).

Project

– Exe Path

–The path location of the selected item.

– Version

–The version number of the selected item.

– File Date

–The date that the selected component was last updated.

Install Options

Type

There are three installation types:

– None

–Use when you don't want other users to be able to install that particular OLE server

–you just want to provide information about it. If you select this option, when users attempt to install the OLE server, they will get an error message alerting them that they cannot install it.

– Copy Server

–Installs a copy of the selected OLE server to the user's local machine and registers it.

– Use Setup App

–When selected, performs the OLE server installation by running a setup application program that you provide. This is useful when the OLE server requires additional files to operate correctly. The setup program performs the installation of the files, putting them in the correct locations on the system.

Path

The Path option is used only when the Use Setup App option is selected. This is the path to the setup application program which is run.

Note The Install Options can be changed only when a component is selected in the main Component Manager window.

Close

Closes the Properties dialog box.

Update

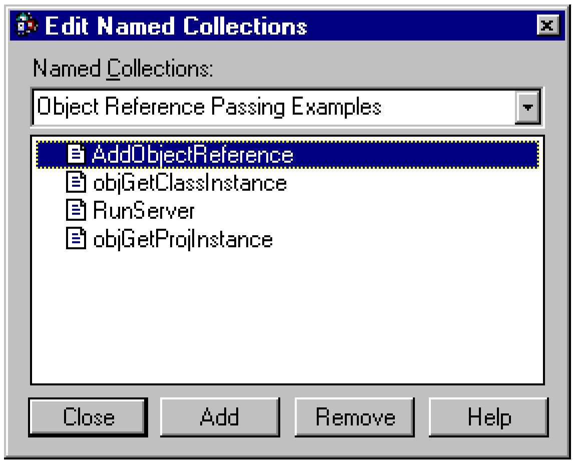
Updates the selected item with any changes made.

See Also

Add OLE Components Dialog Box
Associated Files Tab, Properties Dialog Box
Description Tab, Properties Dialog Box
Switching Between Local and Remote Execution

Edit Named Collections Dialog Box

See Also



Use the Edit Named Collections dialog box to add, edit, or remove items in a collection list.

Tip A good method for editing a collection list is to place the Edit Named Collections dialog box next to the main Component Manager window. You can then drag items from the Results pane and drop them into the Edit Named Collections dialog box.

– To open the Edit Named Collections dialog box, either click the Show Collection Lists toolbar button, or click Show Collection Lists on the View menu. Then, click the right mouse button over the collection you want to edit, and click Edit.

– To close the Edit Named Collections dialog box, click Close or double-click the Control-menu box.

Dialog Box Options

Named Collections

Contains a list of named collections to edit.

Close

Closes the Edit Named Collections dialog box.

Add

Adds the item selected in the Results pane to the selected collection list. Alternatively, you can drag items from the Results pane and drop them into the Named Collections list box.

Note You can add a new collection to the Named Collections list in the Criteria pane by clicking the right mouse button in the Named Collections box displayed, clicking Add, and then specifying a name in the dialog box.

Remove

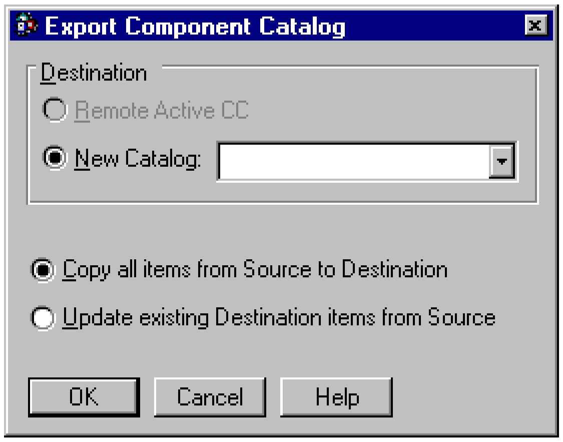
Removes the selected item from the selected collection list.

See Also

Component Manager Toolbar
Overview of the Microsoft Component Manager

Export Component Catalog Dialog Box

See Also



Destination: Remote Active CC [✓ New Catalog:]

[✓ Copy all items from Source to Destination] Update existing Destination items from Source

Use the Export Component Catalog dialog box to synchronize the contents of one component catalog with another. This is generally used to create a portable component catalog for use when you work on a stand-alone computer, such as a laptop, and don't have access to remote catalogs on the network.

If you don't have write permission to the destination component catalog, the operation is halted. If you have write permission to the destination server, but tables with the same names exist in the destination component catalog, you will be prompted to overwrite existing data in the remote catalog.

– To open the Export Component Catalog dialog box, click Export Component Catalog on the Catalog menu, or click the right mouse button in the Scope pane, and then click Export Component Catalog.

– To close the Export Component Catalog dialog box, double-click the Control-menu box. To cancel your settings, click Cancel.

Dialog Box Options

Destination

– Remote Active Catalog

–Exports the selected items to the currently active destination component catalog.

– ODBC DSN

–Exports the selected items to a new component catalog. If a database is not listed in this field, you can add it with the ODBC Control Panel program.

Copy all items from Source to Destination

When selected, all items from the local (source) component catalog are exported to the destination component catalog.

Update existing Destination items from Source

When selected, items in the destination component catalog with the same name as those in the local catalog are updated with the items from the local catalog, synchronizing the data between the two component catalogs.

See Also

Export Component Catalog Command (Catalog Menu)

Finding and Sorting Components

See Also

Once you have defined a component catalog's properties and added and tagged components, you can use the selection lists in the Criteria pane to filter the items shown in the Results pane.

To find components

1. From the View menu, click Show Filter & Sort Criteria.
2. Select the filter choices in the Criteria pane.   For example, in the Sample Components catalog, choosing a Sample Type of Data Access and a Comp Type of Out-Of-Process Server might yield three components: BookSale Server, Jet1Proj, and ODBC1Proj.   This means that these three servers share the same Sample and Comp type.
3. Click the Find Results Now toolbar button.

To sort components

1. Select a non-zero value in the sort selector button (the small button to the right of the filter boxes) of one or more criteria.   To do this, click the button, which defaults to 0, and select a numeric value.
2. Click the Find Results Now toolbar button.

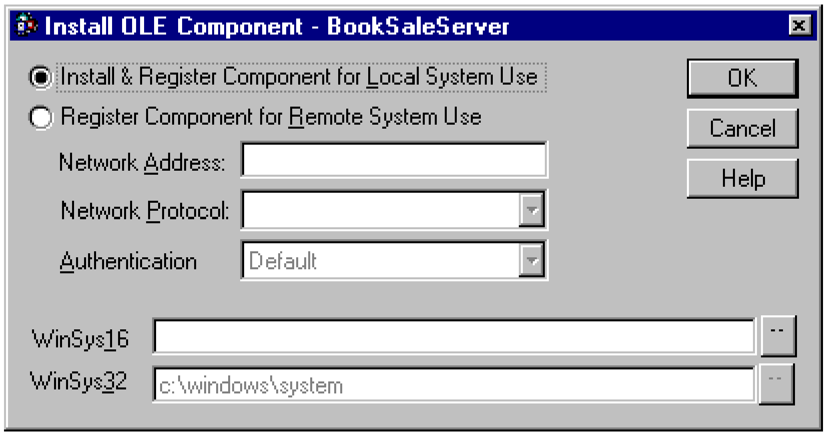
You will see a hierarchical listing of all available components listed under the sorted criteria names.   For example, if you choose 1 for the Sample Type property tag and 2 for the Comp Type property tag, you will see a list of components, listed first under sample types, and then component types.   After that, the components are listed in the usual nested order.

See Also

Add OLE Components Dialog Box
Component Manager Toolbar
Overview of the Microsoft Component Manager
Property Tags Tab, Properties Dialog Box

Install Dialog Box

See Also



(●) Install & Register Component for Local System Use
( ) Register Component for Remote System Use

Network Address:

Network Protocol:

Authentication: Default

WinSys16

WinSys32: c:\windows\system

Use the Install dialog box to specify a local or remote server from which to install the selected OLE component. When you first click the Install command, the Component Manager checks to see if the component has been used remotely before. If so, it loads the previous values for the Network Address, Network Protocol, and Authentication Level as default values.

Note If you are running a Windows 16-bit system, the necessary RPC files needed for remote access must be installed on your local system. Also, while 32-bit servers cannot be installed locally on 16-bit systems, they can be registered for remote use, or when you want to experiment with such servers on your local system.

– To open the Install dialog box, select a component, click the Install command on the Results menu, or click the right mouse button over the Results pane, and then click Install.
– To close the Install dialog box, double-click the Control-menu box.

Dialog Box Options

Install & Register Component for Local System Use

When selected, the remote component is copied to your local system. This option is useful for laptop computers where you may not have access to a remote system.

Clicking OK will perform the installation specified in the Details tab (either Copy .EXE or Use Setup App).

Register Component for Remote System Use

When selected, the component is registered for use on the remote system. Since it will be used on the remote system, it is not copied onto your local system. The network address, protocol, and authentication level are used to locate and use the remote system.

– Network Address
–The network address of the system where the server component is located. You can enter the name of the remote server with or without preceding backslashes (\\). For example, you can enter either \\MYSERVER or MYSERVER.
– Network Protocol
–The network protocol that the client and server systems use to communicate with each other. The network transport protocols supported within the Component Manager are:
– TCP/IP
– SPX
– Named Pipes
– NetBIOS (over NetBEUI, TCP and IPX)
– Datagram (IPX and UDP)

- Authentication

–The RPC (remote procedure call) authentication to use on the client system. Authentication is a tool used to ensure data privacy and integrity, and is useful only if you are concerned about the security of your data.

The authentication levels supported by the Component Manager are:

- Default
- No Authentication
- Connect
- Call Packet
- Integrity Packet
- Privacy Packet

WinSys16

The path to your 16-bit Windows system directory, if you have one installed on your machine. If it is in the same directory as the 32-bit systems directory, then specify the 32-bit system path here.

WinSys32

The path to your 32-bit Windows system directory, if you have one installed on your machine.

See Also

Add OLE Components Dialog Box
Details Tab, Properties Dialog Box
Switching Between Local and Remote Execution

Switching Between Local and Remote Execution

See Also

Once a component has been added to a component catalog and its registry (.VBR) file has been added to the Associated Files list for the classes of an .EXE file, you can use the Component Manager to switch back and forth between local and remote execution of the OLE server.   You can do this with the Install dialog box.

You might elect to do this, for example, if you are testing or debugging a component locally, and you want to switch to testing it remotely.   This is useful in evaluating the performance benefits of running a server locally versus running it remotely, since you can use the Install dialog box to switch between local and remote versions of the same component.   Another scenario might be where an administrator installs all active components on a centralized remote system for use by an entire department, and matching versions of the components are kept on other servers for experimentation or archival purposes.

Note   You can install whole components only; you cannot install individual classes or members from components.

To set a component's local/remote execution state

1. Select a member in the Results pane of the class you want to run.
2. Click the right mouse button over the Results pane and click Install.
3. Select Install & Register Component for Local System Use to copy the component to your local system, or select Register Component for Remote System Use to register the component for use on a remote system.

   Note   The Register Component for Remote System Use option only registers the component for remote access from the local (client) system that the Component Manager is running on. It does not install the component on the remote system, because it is assumed that the administrator of the remote system will be responsible for doing this.

   ---

4. Click OK to accept your settings.

The selected class is now registered as remote or local, depending on which option you select.

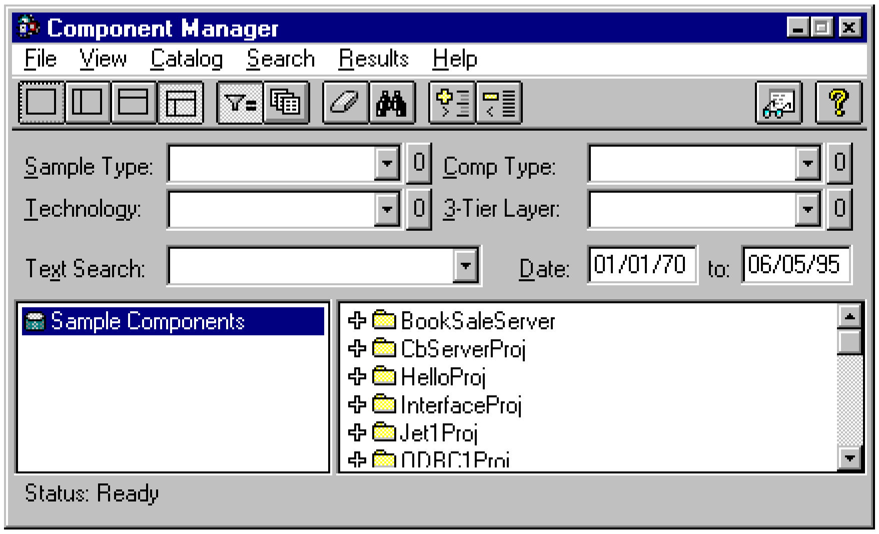
Note   If you want to use the Component Manager rather than the Client Registration utility to set the local/remote state of an OLE server, use the Associated Files tab of the Component's Properties dialog to add the .VBR file for the project or class to the Associated Files tab if it has not already been added.   For information on the Client Registration utility, see Chapter 3, "Client/Server Tools in the Enterprise Edition," of *Building Client/Server Applications with Visual Basic*.

See Also

Associated Files Tab, Properties Dialog Box
Add OLE Components Dialog Box
Overview of the Microsoft Component Manager
Install Dialog Box

Overview of the Microsoft Component Manager

See Also



01/01/70 to 06/05/95

BookSaleServer
CbServerProj
HelloProj
InterfaceProj
Jet1Proj
ODBC1Proj

The Microsoft Component Manager is a tool to help you:

- Locate OLE servers scattered throughout a network.
- Group and catalog related OLE objects.
- Install or register the OLE servers for use in your Visual Basic development projects.
- Provide detailed information about OLE servers.

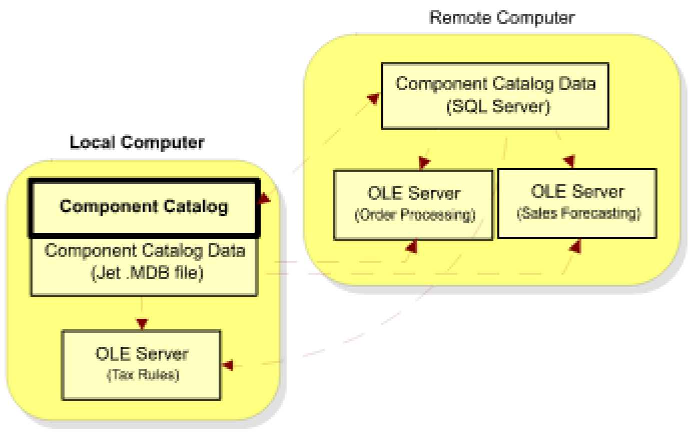
Locating OLE Server Data

The Component Manager provides services for cataloging, locating, and using OLE components in your Visual Basic projects. You can group components into multiple component catalogs that contain variable scopes and cataloging characteristics. Component catalogs can reside on your desktop or in any remote ODBC database.

The Component Manager creates an empty component catalog the first time it is started. This catalog is listed in the Scope pane of the main Component Manager screen. The right pane of the main screen is called the Results pane, which is where components are listed once they have been added to the catalog. The top pane of the main screen is called the Criteria pane, which provides an area for users to specify filter and sort information to help them locate specific components that are in a catalog. (The actual location of the various panes depends on the view layout that you choose.) Clicking the right mouse button over either the Scope pane or the Results pane displays a list of commands that operate on those panes.

OLE servers can be complex, containing many different classes and members. To help you track and logically group information on various components in your enterprise, you can create lists of related OLE servers known as named collections. Using named collections, you can see at a glance which OLE servers are available to you, for example, for preparing tax forms, performing inventory, creating marketing surveys, and so on. Alternatively, you can use a filtering system to sort through OLE server data and display it in the manner you want.

You can also track both local and remote components using the Component Manager. The following diagram illustrates the relationship between local and remote components and component catalogs. Component catalog data can reside on local or remote systems, as can OLE servers. Local component catalogs can reference remote components, such as the Order Processing and Sales Forecasting servers on the remote system, or a local component, such as Tax Rules.


Remote Computer
Component Catalog Data (SQL Server)
OLE Server (Order Processing)
OLE Server (Sales Forecasting)
Local Computer
Component Catalog
Component Catalog Data (Jet .MDB file)
OLE Server (Tax Rules)

Grouping Related OLE Objects

Using the Component Manager, you can create tags that relate to particular services or groups of services, such as OLE objects that forecast sales or process orders. You then associate these tags with the individual OLE objects that perform those particular services. Once this is complete, you can search for and use services based on tags in your project. For example, tax rules, order processing components, sales forecasting formulas, and so on.

The Component Manager does not store the actual OLE servers themselves. Instead, it stores references to and information about existing OLE servers in a database known as a component catalog. By browsing a component catalog, you can view the hierarchy of OLE components, the classes they contain, and the members contained in each class. You can also browse other component catalogs on any remote ODBC database.

Installing and Registering OLE Servers

Users of your component catalog can view information on OLE servers, or even install them for use in their projects. The End User Installation options allow you to determine if and how each component will be installed if users attempt to install it locally on their machines.

When you want to install a component, you can install and register components for local use, or register the component for use on a remote system with specific network address, protocol, and authentication level values.

Providing Detailed Information About OLE Servers

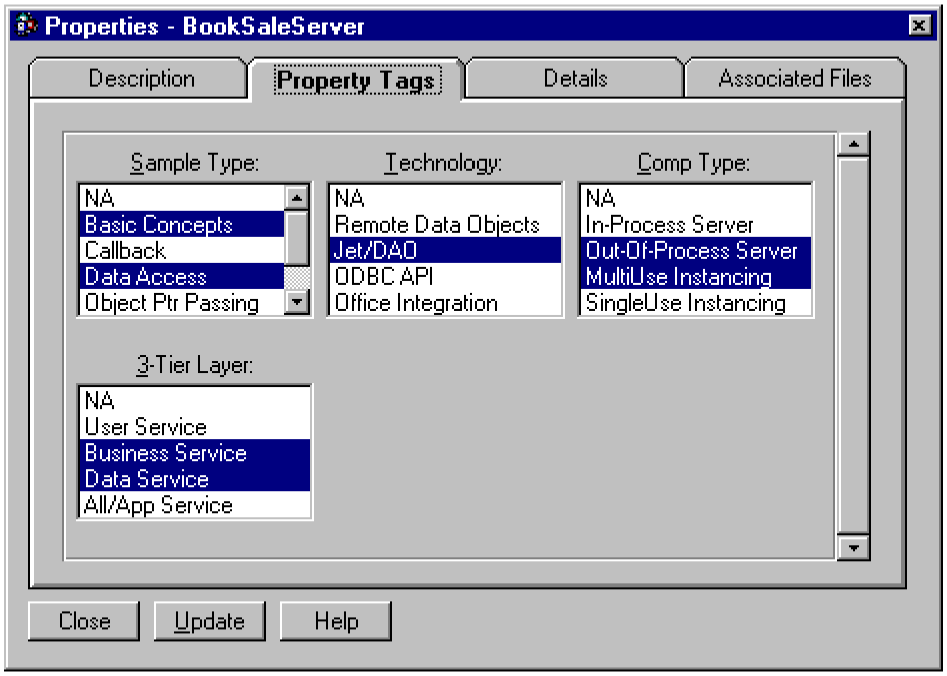
You can associate files that contain detailed information relating to the various OLE servers: how to use them, sample client code, registration files, specifications, licensing information, and so forth. The Component Manager tracks OLE servers located on other systems on the network, as well as OLE servers located on the machine on which it is installed.

See Also

Adding Components to a Component Catalog
Adding Information About Components
Associated Files Tab, Properties Dialog Box
Customizing a Component Catalog
Finding and Sorting Components
Install Dialog Box
Switching Between Local and Remote Execution

Property Tags Tab, Properties Dialog Box

See Also



Use the Property Tags tab to select global information relating to all classes in a component, or all members in a class, by setting the properties on the component and class, respectively.

– To open the Property Tags tab, select an item in the Results pane, and click the right mouse button. Click Properties, and then click the Property Tags tab.

– To close the Property Tags tab, click the Close button or double-click the Control-menu box.

Tab Options

The scrollable list boxes list all the available properties of a component catalog and their enumerated values, and indicates which values apply to the selected item in the main Component Manager window. When you change the tag for an item, the tags for all items subordinate to it are also changed.

Close

Closes the component Properties dialog box.

Update

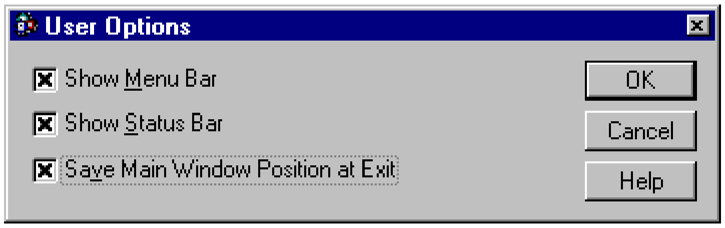
Updates the component with the currently selected values.

See Also

Customizing a Component Catalog
Finding and Sorting Components
Overview of the Microsoft Component Manager
Tagging Components

User Options Dialog Box

See Also



[✓ Show Menu Bar]
[✓ Show Status Bar]
[✓ Save Main Window Position at Exit]

Use the User Options dialog box to show or hide the menu and status bars, and choose whether or not to save the main window's position at exit.

– To open the User Options dialog box, click the User Options toolbar button, or click User Options on the View menu.

– To close the User Options dialog box, click the Close button or double-click the Control-menu box.

Dialog Box Options

Show Menu Bar

Allows you to show or hide the menu bar.

Show Status Bar

Allows you to show or hide the status bar.

Save Main Window Position at Exit

Allows you to save the size and position of the main Component Manager window when you exit.

See Also

Component Manager Toolbar

collection

A named group of related components. For example, a collection named Tax Preparation Objects might contain the names of OLE objects like EndOfYear, RoyaltyCalc, and ExemptionCalc.

collection list

A list of named groups of related collections. For example, Tested Components might be a list of all components that have been tested.

component

Any software that supports OLE Automation, which means it can be used programmatically in a custom solution.   This includes OLE controls (.OCX files), Visual Basic-based OLE Automation servers, and Visual C-based OLE Automation servers.

component catalog

A sharable database of information that describes and manages components−generally OLE servers.   A component catalog does not contain the objects themselves, but contains references to where the objects reside on a computer or network.

Criteria pane

Depending on how you arrange the panes, the Criteria pane is generally the top-most region of the Component Manager window, used for defining the criteria for displaying data in the Scope and Results panes.

OLE server

An application that provides objects to other applications.

Results pane

Depending on how you arrange the panes, the Results pane is generally the lower right-most region of the Component Manager window, used for displaying the components contained in the component catalog selected in the Scope pane, according to the criteria defined in the Criteria pane.

Scope pane

Depending on how you arrange the panes, the Scope pane is generally the lower left-most region of the Component Manager window which displays the available added component catalogs.

tag

An association between a component and the property values of a catalog.   For example, a component named TaxPreparation might have the Royalty Calculations search tag value for the Calculation property.

Control-menu box

In Microsoft Windows, the box in the upper-left corner of a window that opens the Control menu.

For applications, this menu displays commands that allow you to restore, move, size, minimize, maximize, or close the current application.   In addition, you may be able to switch to or run other Microsoft Windows-based applications.

For windows within an application, this menu displays commands that allow you to size, move, split, or close the active window.

Tagging Components

See Also

As your component catalogs grow, property and value tags can help you sort through and find components quicker. The properties are displayed in the Criteria pane of the main Component Manager window. Once the properties and values for a catalog have been defined, individual components can be tagged with the appropriate value(s).

To tag components

1. Select a component in the Results pane.
2. Click the right mouse button, and then click Properties.
3. Click the Property Tags tab.
4. Select all property values that apply to the selected component.

Tip To more easily set properties for several components, leave the Properties dialog box open and next to the main Component Manager window. The Properties dialog updates each time you select a new component in the main Component Manager window.

See Also

Customizing a Component Catalog
Property Tags Tab, Properties Dialog Box
Overview of the Microsoft Component Manager

An error occurred trying to enumerate the Windows registration database. The database may be corrupted.

Component Manager was unable to list the components available as OLE servers from the Windows Registry because the registration data on your hard disk may need to be repaired.

The Windows Registry registration database contains references to components that may be used as OLE servers. Applications that include OLE components add these references to the Windows Registry when they are installed or loaded for the first time. If the registration database needs repair, this information will be unreadable by Component Manager.

For information on restoring the registration database, see your Windows documentation or online Help (accessible by pressing F1 from within the Registration Editor).

An error occurred trying to open the Windows registration database.

Component Manager was unable to open the Windows Registry to find information on components available for use as OLE servers.

The Windows Registry registration database contains references to components that may be used as OLE servers. Applications that include OLE components add these references to the Windows Registry when they are installed or loaded for the first time. If the registration database needs repair, this information will be unreadable by Component Manager.

For information on restoring the registration database, see your Windows documentation or online Help (accessible by pressing F1 from within the Registration Editor).

An error occurred trying to query the Windows registration database. The database may be corrupted.

Component Manager was unable retrieve information on components available as OLE servers because the Windows Registry data on your hard disk may need to be repaired.

The Windows Registry registration database contains references to components that may be used as OLE servers.   Applications that include OLE components add these references to the Windows Registry when they are installed or loaded for the first time.   If the registration database needs to be repaired, this information will be unreadable by Component Manager.

For information on restoring the registration database, see your Windows documentation or online Help (accessible by pressing F1 from within the Registration Editor).

Application requires Microsoft Windows 32-bit extensions.

Component Manager could not load the file you selected because its application is incompatible with the operating system.

If the computer you are working with uses a 16-bit operating system (such as Windows 3.1 or Windows for Workgroups 3.11), check to see if the application was designed for a 32-bit Windows operating system (such as Windows NT or Windows 95). If this is the case, replace the application with a 16-bit version, move it to a 32-bit operating system, or change its association so that it is associated with an application that will run on your operating system.

Application was designed for a different operating system.

The file could not be loaded because it was designed for a different operating system. Replace the application with a version compatible with your operating system, or change the file's association so that it is associated with an application that will run on your operating system.

Application was designed for MS-DOS 4.0.

The file could not be loaded because it was designed for an older version of MS-DOS. Replace it with a version compatible with your operating system, or change the file's association so that it is associated with an application that will run on your operating system.

Attempt was made to dynamically link to a task, or there was a sharing or network protection error.

The file could not be loaded because it may already be running.

To run two instances of the same file, you must have loaded in memory a program that allows multiple instances of the same file, such as SHARE.EXE, or use an operating system that allows sharing implicitly, such as Windows NT or Windows 95.

Attempt was made to load a compressed executable file. The file must be decompressed before it can be loaded.

The file could not be loaded because it is compressed.

You will need to decompress the file using a file decompression utility before it can be loaded.   See the file's provider to learn which utility can be used to decompress it.

Attempt was made to load a real-mode application (developed for an earlier version of Windows).

The application could not be loaded because it was designed for an earlier version of Windows. Replace the application with a version compatible with your operating system, or change the file's association so that it is associated with an application that will run on your operating system.

Attempt was made to load a second instance of an executable file.

The executable file may contain multiple data segments that were not marked read-only. The application could not be loaded because it may use files that are currently being used by another application. Close the previous instance of the application you tried to load, and then try again.

Could not install component.

See Also

There are three possible causes for this error:

– Installing the component you selected requires a setup file that could not be found. The Add OLE Components dialog box allows you to specify a setup application to be used when the component is installed. Installation may fail if the component requires this application, and the application's path and filename are not specified, or are not valid. Ensure that the path and filename for the setup application are correct.

– Component Manager is unable to read information about the component from the component catalog database because its reference to the component may be invalid. Delete the component from the catalog, and add it again to update the reference.

– Component Manager is unable to read information about the component from the component catalog database because the database may need to be repaired. If you have made a backup of the database, use the backup to restore it. If you have no backup copy, it may be necessary to recreate the component catalog.

See Also

[Add OLE Components Dialog Box](#)
[Install Dialog Box](#)

Could not install component: must have .REG file or .VBR file.

See Also

Remote installation of a component requires that a .REG file or .VBR file be listed in the Associated Files tab of the Properties dialog box.

When you add a new component, its corresponding Visual Basic registration (.VBR) file or Windows registration (.REG) file is automatically added to the list in the Associated Files tab if they exist at the same path as the component file.   When you install a component for remote use, these files are used to update the Windows Registry to include references to the Remote Automation proxy and the OLE component's network address.

If a reference to the needed file is not included in the list, you can add it by clicking the Add button in the Associated Files tab and selecting the appropriate .VBR file.

See Also

Associated Files Tab, Properties Dialog Box

Install Dialog Box

Could not install component: network address not specified.

See Also

You did not include the network address of the system where the OLE component is located.

The network address, protocol, and authentication level are used to locate the remote system.   The network address field specifies the system where the server component is located.

Type the appropriate network address in the Network Address field of the Install OLE Component dialog box.

See Also

Install Dialog Box

Could not install component: network protocol not specified.

See Also

You did not include the network protocol that the client and server systems use.

The network protocol, address, and authentication level are used to locate the remote system.   The Network Protocol (TCP/IP or Named Pipes, for example) field specifies the protocol your client/server system uses for communication.

Select the appropriate network protocol from options in the Network Address field in the Install OLE Component dialog box.

See Also

Install Dialog Box

Could not open Type Library.

Component Manager was unable to get information about available OLE servers from the type library (TypeLib).

The file you selected may not be a valid TypeLib file, or the TypeLib file you selected may need to be repaired.   Restore the file from a backup copy, or reinstall the application associated with that type library.

Dynamic link library (DLL) file was invalid. One of the DLLs required to run this application was corrupt.

The application could not be loaded because a .DLL file it requires may need to be repaired. Restore the .DLL file from a backup copy, or reinstall the application that uses the DLL.

Error creating directory: 'directory'.

The component could not be installed because installation requires that certain files be copied to an OLE server directory of your Windows or <Common Files> directory, and this directory could not be created.   Make sure that directories used during installation are not write-protected.

Error removing file from Catalog

Component Manager was unable to remove the associated file from the component catalog's database.   (When this error occurs, it is preceded by a message with more specific information.)

References to associated files in the database may need to be repaired.   To update these references, exit and restart Component Manager.

Error retrieving type information.

Component Manager was unable to get information about available OLE components from a type library (TypeLib) or OLE server. This can occur for two reasons:

– The TypeLib or OLE server file you selected may need to be repaired. Restore the file from a backup copy, or reinstall the application that uses it.

– The file you selected to add does not contain createable object classes. Not all .EXE files, for example, are valid OLE servers. If you are unsuccessful in adding references to components using an application's .EXE file, look instead for a corresponding TypeLib (.TLB, .OLB) file.

Executable file was invalid. Either it was not a Windows application or there was an error in the .EXE image.

The file could not be loaded because it is damaged or is not a Windows-based application. If the file is a Windows-based application, it may need to be repaired. If this is the case, restore the file from a backup copy, or reinstall the application.

File was not found.

The file you selected could not be loaded because it could not be found in the path specified.   Verify that the path and filename are correct.

Installation Failure. Please specify a valid path to the Windows System directory.

See Also

You did not specify the correct Windows System directory path in one or both of the WinSys fields in the Install OLE Component dialog box.

The WinSysxx fields instruct the Component Manager where to find the Remote Automation support files.   To be able to use OLE servers from both 16- and 32-bit applications, you must provide the paths for both of your operating systems.

WinSys16 specifies the path to your 16-bit Windows system directory, if you have one installed on your machine.

WinSys32 specifies the path to your 32-bit Windows system directory, if you have one installed on your machine.

See Also

Install Dialog Box

Installation Failure. Unable to copy file.

See Also

Installation failed because Component Manager was unable to copy a necessary file to its appropriate directory.

Files copied during component installation (.VBR, .REG, and/or .TLB files) are listed in the Associated Files tab of Component Manager's Properties dialog box. There are two possible reasons why these wouldn't be copied:

– The needed file or its destination path (the Windows or <Common Files> directory) may already exist as write-protected, and can't be overwritten. Remove write protection from the file or directory, or delete the existing file, and then install the component.

– The needed file may already exist in the directory as a write-protected file, and can't be overwritten. Delete the existing file, and then install the component.

– The file reference is no longer valid. Check to be sure that referenced files still exist in the paths given. If not, click the Add button in the Associated Files dialog box to add a reference to the file.

See Also

Associated Files Tab, Properties Dialog Box

Install Dialog Box

Library required separate data segments for each task.

The file could not be loaded because it was designed for a different operating system. Replace the application with a version compatible with your operating system, or change the file's association so that it is associated with an application that will run on your operating system.

Maximum limit reached, unable to create additional items.

See Also

You can specify up to 40 property values for each property label.

See Also

Catalog Properties Dialog Box

Path was not found.

The file could not be loaded because the path specified could not be found.   Check to be sure the path and filename are correct.

Please select a component before using this command.

You did not select a component name before using this command.   Click the name of the component to select it, and then choose the command.

Please specify a title for the new catalog

See Also

You did not include any text in the Title field of the Create New Catalog dialog box.   When creating a new component catalog, you must give it a title. Component catalog titles can include any text and be as many as 50 characters long.

See Also

Create New Catalog Dialog Box

Please specify a valid file name

See Also

You typed an invalid filename or path in the FileName field of the Create New Catalog dialog box.   Check the path and filename to be sure you have typed them correctly.   For example, the following characters are not valid within filenames:

:   "   <   > | *   ?   \   /

On Win32 systems (including Windows 95 and Windows NT), filenames may include spaces and may be longer than eight characters.

See Also

Create New Catalog Dialog Box

Property limit reached, unable to create additional properties

See Also

Each component catalog may include up to eight user-defined properties.

See Also

Catalog Properties Dialog Box

Selected file does not contain appropriate OLE typeinfo data. Unable to add component

The file you selected can't be added to the component catalog because it doesn't contain information about OLE components.

Files containing OLE components include executable (.EXE, .DLL) or TypeLib (.TLB, .OLB) files. Even so, not all .EXE files are valid OLE servers.   To reduce the size of .EXE files, some applications provide OLE servers that store these components in TypeLib files.

System was out of memory, executable file was corrupt, or relocations were invalid.

The file could not be loaded. There are two possible reasons for this error:

– Its application is corrupted. Replace the application file with a backup, or reinstall it.
– There are insufficient system resources. Reboot the computer and try loading the file again.

There are no browseable objects registered in your Windows system.

The Windows Registry currently contains no objects which may be added to the component catalog.

Applications containing OLE components are registered in the Windows Registry when they are installed or loaded for the first time.

You have not installed any applications with OLE components; the Windows Registry will not contain references to any.

For information on restoring the registration database, see your Windows documentation or online Help (accessible by pressing F1 from within the Registration Editor).

There was insufficient memory to start the application.

The application could not be started because there is insufficient system memory available.
Close other applications to free up memory.

This file cannot be opened.

Component Manager could not open the file you selected. This error can occur for three reasons:

– The file is not associated with an application that can be used to open or edit it. See your Windows documentation for instructions on associating files with applications.

– The component has been moved. To update its path and name reference in the component catalog, use Component Manager to add the component again.

– A file used by Component Manager has been damaged. To restore the file, reinstall Component Manager.

This is not an installable component.

The component you selected is not designed for installation as an OLE component.

The component does not contain type information, or information describing the objects and interfaces it exposes.   If the component you selected is an .EXE or .DLL file, there may be a corresponding type library (TypeLib) file that does contain the needed information. These files will have an .OLB or .TLB extension and may be located in the same directory as the .EXE or .DLL, or in the Windows System directory.

Type of executable file was unknown.

The file could not be loaded because it may have been designed for a different operating system.   Replace the application with a version compatible with your operating system, or change the file's association so that it is associated with an application that will run on your operating system.

Unable to create new catalog

Component Manager was unable to create a new component catalog. There are three possible reasons for this error:

– There is insufficient system memory available. Close other applications to free up memory.

– You have created the maximum allowable number of component catalogs. Remove a catalog and try again.

– If this error was related to the component catalog database, a more specific error message preceded this one.

Unable to find topic

Component Manager was unable to find a reference to the file you selected in the component catalog's database.

Close and reopen Component Manager to update its references.   If this error occurs again, try updating the reference to the file by removing it from the catalog, and then adding it again.

If this fails, your component catalog's database (.MDB file) may need repair, and you should restore it from a backup copy.

Unable to load Component Catalog.

Component Manager was unable to load the component catalog database file. This error can occur for three reasons:

- There isn't enough available system memory. Closing another application may free up enough memory to allow the component catalog to be opened.
- The maximum number of component catalogs may already be open. Close or delete a component catalog before opening a new one.
- The file you tried to view can't be used as a component catalog. Component Manager searches for specific characteristics to identify databases as component catalogs. The database you selected either does not have these characteristics, or its data source name (DSN) is not included among the data sources accessible from your computer. Component Manager opens a connection to a component catalog database through a reference to its data source name. To update this reference, use the ODBC Administrator utility installed with Visual Basic to add the data source name corresponding to the database of the component catalog you want to open.

Unable to retrieve information for this method or property; code template may be missing parameters.

Component Manager was unable to find the information it needs to build a reference to the component's methods and properties.

The component's type library (TypeLib), which contains references to its methods and properties, may be invalid or in need of repair.   If it needs repair, reinstalling the component may create a usable version of the file on the hard disk.   If it is invalid, obtain an updated version of the file from the vendor/author.

Unable to set an active catalog. Closing.

Component Manager was unable to specify an .MDB file as an active component catalog.

Component Manager must be able to set a currently active component catalog to run. By default, it creates a new component catalog if no other exists to set as active. Run Component Manager again to create a new active component catalog. If this fails, delete or rename the Component Manager initialization file (\WINDOWS\CMPMGR.INI), and then try again.

Windows version was incorrect.

The file could not be loaded because it requires a different version of Windows.   Obtain a copy of the file that is compatible with your version of Windows.