Timesheets MTS Software - Visual Basic 6 ADO Tutorial

Scope

This tutorial aims to provide an intermediate Visual Basic 6 user with the basic skills required to retrieve and manipulate data from many commercial databases and any ODBC compliant database. It assumes that those reading it have an understanding of the Object Oriented methodology of Visual Basic 6, and also of the typical programming structures found in Visual Basic. For example, no explanation of what a method is will be given, and it will be assumed that readers will know how the For Each Obj in Collection structure works.

Finding this tutorial helpful? Why not

 to help cover the bandwidth it chews up? It's the most popular page on this site bar none!

Introduction

Visual Basic 6 obsoletes the previously used database access technology provided by Jet and provides a new one known as ADO or Active Data Objects. This technology allows users to access data easily from many existing databases (such as Access or Paradox) or from ODBC compliant databases like Oracle or MS SQL Server. Using ADO is quite simple and allows programmers to provide flexible database front ends to users that are reliable and include many features. As a VB6 programmer ADO is important to know as most commercial VB programming exercises involve providing a front end to some sort of database.

Following are some of the key objects found in the ADO object model and some of their key methods and properties.

Connection Object

This object represents an open connection to the data source. This connection can be a local connection (say App.Path) or can be across a network in a client server application. Some of the methods and properties of this object are not available depending on the type of data source connected to.

Key Properties

Name	Data Type	Description
ConnectionString	String	Defines in string form the location of the data source you wish to connect to. Key fields in the string you will use are the "Provider=" and the "Data Source=" fields. You may also use the "Mode=" field. See descriptions of those properties for a more detailed view of each one. Some examples of connection strings follow: **Data Source=c:\test.mdb;Mode=Read|Write;Persist** - Connects to an Access database with read/write/persist permissions **driver={SQL Server) server=bigsmile;uid=sa;pwd=pwd;database=pubs** - Connects to an SQL Server database called bigsmile as user sa with password pwd to database pubs.
Provider	String	A string defining the provider of a connection object. An example follows:

		Provider=Microsoft.Jet.OLEDB.4.0 -This string connects to a MS Jet 4.0 compliant database (an Access DB for example.)
Mode	connectModeEnum	Sets (or returns) the permissions for modifying data across the open connection. Available permissions include Read, Write, Read/Write, and various Share/Deny types.
CursorLocation	cursorLocationEnum	Sets the location of the cursor engine. You can select client side or server side, for simpler databases (like Access) client side is the only choice.
ConnectionTimeout	Long	Time in seconds that a connection will attempt to be opened before an error is generated.
CommandTimeout	Long	Time in seconds that an command will attempt to be executed before an error is generated.

Key Methods

Name	Description
Close	Closes an open connection.
Open	Opens a connection with the settings in the ConnectionString property.

Command Object

A command object specifies a specific method you intend to execute on or against the data source accessed by an open connection.

Key Properties

Name	Data Type	Description
ActiveConnection	conConnection as ADODB.Connection	Defines the Connection object the command belongs to.
CommandText	String	Contains the text of the command you want to execute against a data source. This can be a table name, a valid SQL string, or the name of a stored procedure in the data source. Which one you use is determined by the CommandType property.
CommandType	commandTypeEnum	Defines the type of the command. The three most commonly

		used would be adCmdText, adCmdTable, and adCmdStoredProc. The setting adCmdText causes the CommandText string to be evaluated as an SQL string. The setting adCmdTable causes the CommandText string to be evaluated as a table name and will return the all of the records and columns in a table. The setting adCmdStoredProc causes the CommandText string to be evaluated as a stored procedure in the data source.

Key Methods

Name	Description
Execute	Executes the query, SQL statement, or stored procedure specified in the CommandText property and returns a RecordSet object.

RecordSet Object

The RecordSet object represents a complete set of records from an executed command or from an underlying base table in the database. A key thing to note is that a RecordSet object references only one record at a time as the current record.

Key Properties

Name	Data Type	Description
CursorLocation	CursorLocationEnum	This sets or returns the location of the cursor engine for the database. The options are client side and server side, for less advanced databases like Access you may only be able to select client side.
CursorType	CursorTypeEnum	Sets the cursor type. CursorType typically determines when and to whom database changes are immediately visible to. For client side cursor locations only one type of CursorType is available, adOpenStatic.
EOF and BOF	Boolean	End Of File and Beginning Of

		File. When at the first record of an open RecordSet BOF will be true, when at the last EOF will be true. If you ever return a RecordSet with no records then EOF and BOF will be true. This provides an ideal way of testing if any records have been returned by a query.
Fields	Collection	Returns a collection of the field objects for an open record set. Database fields can be accessed by their name using the RecordSet!FieldName schema, by their index RecordSet.Fields(intIndex) or sometimes by their name from the fields collection RecordSet.Fields("FieldName"). I find in the situation where you know a database structure that the RecordSet!FieldName method is best, where structure is not known, then the RecordSet.Fields(intIndex) may be best.
LockType	LockTypeEnum	sets the lock type on records when they are open for editing. If a client side cursor location is selected then only optimistic and batch optimistic locking are available.
RecordCount	Long	Returns the number of records in an open RecordSet. If for some reason ADO cannot determine the number of records then this will be set to -1.

Key Methods

Name	Description
AddNew	Sets up an open record set to add a new record, once the required values have been set call the Update (or UpdateBatch) method to commit the changes.
Close	Closes an open RecordSet object, make sure that any changes are committed using the Update (or UpdateBatch) method before closing or an error will be generated.
MoveNext	Causes an open RecordSet object to move to the next record in the collection, if the current record is the last record then EOF is set to true.

MoveFirst	Causes an open RecordSet object to move to the first record in the collection.
MoveLast	Causes an open RecordSet object to move to the last record in the collection.
Open	Opens the RecordSet, typically this is done with a valid Command object that includes the command text (or SQL) to get the records you want.
Update	Causes any changes made to the current record to be written to disk.
UpdateBatch	Causes any changes you have made in batch mode to be written to disk. The way to use batch updates is to open the RecordSet object with the LockType adLockBatchOptimistic.

Putting It All Together

I like to think of using ADO as a three step process

- Define and open a Connection to a data source
- Decide what data you need from the data source and define Command objects that will retrieve this data.
- Retrieve the data you require by executing the command objects and manipulate the data using RecordSet objects.

To start using ADO create a new Standard EXE project in your VB6 environment. You'll need to add a reference to the ADO library using the Project -> References menu and selecting the "Microsoft ActiveX Data Objects 2.5 Library" or the "Microsoft ActiveX Data Objects 2.6 Library". Then on the form add a single command button and add the code below to the click event of the button.

Save the project and then open Microsoft Access and create a new database called "database.mdb" in the directory where you have saved your VB project. Add a table called "tabTestTable" to the database. Add two columns to this table with a text data type, the column names do not matter. Add a couple of records to the table with some values in the columns.

Then you can safely run your project and see the results.

If you dont want to do this I have created a VB project that includes all of this that can be downloaded. Just unzip this to a directory and open the project in Visual Studio and try it out.

Got questions? Feel free to ask me at questions@timesheetsmts.com Why not

to help cover the bandwidth this page chews up and you'll be much more likely to get an answer!

```
Private Sub Command1_Click()

  'Define the three objects that we need,
  '   A Connection Object - connects to our data source
  '   A Command Object - defines what data to get from the data source
  '   A RecordSet Object - stores the data we get from our data source

  Dim conConnection As New ADODB.Connection
  Dim cmdCommand As New ADODB.Command
  Dim rstRecordSet As New ADODB.Recordset


  'Defines the connection string for the Connection.  Here we have used fields
  'Provider, Data Source and Mode to assign values to the properties
  ' conConnection.Provider and conConnection.Mode
```

```
conConnection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & _
    App.Path & "\" & "database.mdb;Mode=Read|Write"


'Define the location of the cursor engine, in this case we are opening an Access database
'and adUseClient is our only choice.

conConnection.CursorLocation = adUseClient


'Opens our connection using the password "Admin" to access the database.  If there was no password
'protection on the database this field could be left out.

conConnection.Open


'Defines our command object

' .ActiveConnection tells the command to use our newly created command object.
' .CommandText tells the command how to get the data, in this case the command
'              will evaluate the text as an SQL string and we will return all
'              records from a table called tabTestTable
' .CommandType tells the command to evaluate the .CommandText property as an SQL string.

With cmdCommand
  .ActiveConnection = conConnection
  .CommandText = "SELECT * FROM tabTestTable;"
  .CommandType = adCmdText
End With

'Defines our RecordSet object.

'  .CursorType sets a static cursor, the only choice for a client side cursor
'  .CursorLocation sets a client side cursor, the only choice for an Access database
'  .LockType sets an optimistic lock type
'  .Open executes the cmdCommand object against the data source and stores the
'        returned records in our RecordSet object.

With rstRecordSet
  .CursorType = adOpenStatic
  .CursorLocation = adUseClient
  .LockType = adLockOptimistic
  .Open cmdCommand
End With

'Firstly test to see if any records have been returned, if some have been returned then
'the .EOF property of the RecordSet will be false, if none have been returned then the
'property will be true.

If rstRecordSet.EOF = False Then

  'Move to the first record

  rstRecordSet.MoveFirst

  'Lets move through the records one at a time until we reach the last record
  'and print out the values of each field

  Do

    'Access the field values using the fields collection and print them to a message box.
    'In this case I do not know what you might call the columns in your database so this
    'is the safest way to do it.  If I did know the names of the columns in your table
```

```
    'and they were called "Column1" and "Column2" I could reference their values using:

    '  rstRecordSet!Column1
    '  rstRecordSet!Column2


    MsgBox "Record " & rstRecordSet.AbsolutePosition & " " & _
        rstRecordSet.Fields(0).Name & "=" & rstRecordSet.Fields(0) & " " & _
        rstRecordSet.Fields(1).Name & "=" & rstRecordSet.Fields(1)

    'Move to the next record

    rstRecordSet.MoveNext
  Loop Until rstRecordSet.EOF = True

  'Add a new record

  With rstRecordSet
    .AddNew
    .Fields(0) = "New"
    .Fields(1) = "Record"
    .Update
  End With

  'Move back to the first record and delete it

  rstRecordSet.MoveFirst
  rstRecordSet.Delete
  rstRecordSet.Update


  'Close the recordset

  rstRecordSet.Close
 Else
  MsgBox "No records were returned using the query " & cmdCommand.CommandText
 End If

 'Close the connection

 conConnection.Close

 'Release your variable references

 Set conConnection = Nothing
 Set cmdCommand = Nothing
 Set rstRecordSet = Nothing
End Sub
```