Chapter 37: Working with ADO

Section 37.1: Making a connection to a data source

The first step in accessing a data source via ADO is creating an ADO `Connection` object. This is typically done using a connection string to specify the data source parameters, although it is also possible to open a DSN connection by passing the DSN, user ID, and password to the `.Open` method.

Note that a DSN is not required to connect to a data source via ADO - any data source that has an ODBC provider can be connected to with the appropriate connection string. While specific connection strings for different providers are outside of the scope of this topic, ConnectionStrings.com is an excellent reference for finding the appropriate string for your provider.

```
Const SomeDSN As String = "DSN=SomeDSN;Uid=UserName;Pwd=MyPassword;"

Public Sub Example()
    Dim database As ADODB.Connection
    Set database = OpenDatabaseConnection(SomeDSN)
    If Not database Is Nothing Then
        '... Do work.
        database.Close          'Make sure to close all database connections.
    End If
End Sub

Public Function OpenDatabaseConnection(ConnString As String) As ADODB.Connection
    On Error GoTo Handler
    Dim database As ADODB.Connection
    Set database = New ADODB.Connection

    With database
        .ConnectionString = ConnString
        .ConnectionTimeout = 10             'Value is given in seconds.
        .Open
    End With

    OpenDatabaseConnection = database

    Exit Function
Handler:
    Debug.Print "Database connection failed. Check your connection string."
End Function
```

Note that the database password is included in the connection string in the example above only for the sake of clarity. Best practices would dictate ***not*** storing database passwords in code. This can be accomplished by taking the password via user input or using Windows authentication.

Section 37.2: Creating parameterized commands

Any time SQL executed through an ADO connection needs to contain user input, it is considered best practice to parameterize it in order to minimize the chance of SQL injection. This method is also more readable than long concatenations and facilitates more robust and maintainable code (i.e. by using a function that returns an array of `Parameter`).

In standard ODBC syntax, parameters are given ? "placeholders" in the query text, and then parameters are appended to the `Command` in the same order that they appear in the query.

Note that the example below uses the `OpenDatabaseConnection` function from the Making a connection to a data source for brevity.

```
Public Sub UpdateTheFoos()
    On Error GoTo Handler
    Dim database As ADODB.Connection
    Set database = OpenDatabaseConnection(SomeDSN)

    If Not database Is Nothing Then
        Dim update As ADODB.Command
        Set update = New ADODB.Command
        'Build the command to pass to the data source.
        With update
            .ActiveConnection = database
            .CommandText = "UPDATE Table SET Foo = ? WHERE Bar = ?"
            .CommandType = adCmdText

            'Create the parameters.
            Dim fooValue As ADODB.Parameter
            Set fooValue = .CreateParameter("FooValue", adNumeric, adParamInput)
            fooValue.Value = 42

            Dim condition As ADODB.Parameter
            Set condition = .CreateParameter("Condition", adBSTR, adParamInput)
            condition.Value = "Bar"

            'Add the parameters to the Command
            .Parameters.Append fooValue
            .Parameters.Append condition
            .Execute
        End With
    End If
CleanExit:
    If Not database Is Nothing And database.State = adStateOpen Then
        database.Close
    End If
    Exit Sub
Handler:
    Debug.Print "Error " & Err.Number & ": " & Err.Description
    Resume CleanExit
End Sub
```

Note: The example above demonstrates a parameterized UPDATE statement, but any SQL statement can be given parameters.

Section 37.3: Retrieving records with a query

Queries can be performed in two ways, both of which return an ADO `Recordset` object which is a collection of returned rows. Note that both of the examples below use the `OpenDatabaseConnection` function from the Making a connection to a data source example for the purpose of brevity. Remember that the syntax of the SQL passed to the data source is provider specific.

The first method is to pass the SQL statement directly to the Connection object, and is the easiest method for executing simple queries:

```
Public Sub DisplayDistinctItems()
    On Error GoTo Handler
    Dim database As ADODB.Connection
    Set database = OpenDatabaseConnection(SomeDSN)
```

```
    If Not database Is Nothing Then
        Dim records As ADODB.Recordset
        Set records = database.Execute("SELECT DISTINCT Item FROM Table")
        'Loop through the returned Recordset.
        Do While Not records.EOF      'EOF is false when there are more records.
            'Individual fields are indexed either by name or 0 based ordinal.
            'Note that this is using the default .Fields member of the Recordset.
            Debug.Print records("Item")
            'Move to the next record.
            records.MoveNext
        Loop
    End If
CleanExit:
    If Not records Is Nothing Then records.Close
    If Not database Is Nothing And database.State = adStateOpen Then
        database.Close
    End If
    Exit Sub
Handler:
    Debug.Print "Error " & Err.Number & ": " & Err.Description
    Resume CleanExit
End Sub
```

The second method is to create an ADO `Command` object for the query you want to execute. This requires a little more code, but is necessary in order to use parametrized queries:

```
Public Sub DisplayDistinctItems()
    On Error GoTo Handler
    Dim database As ADODB.Connection
    Set database = OpenDatabaseConnection(SomeDSN)

    If Not database Is Nothing Then
        Dim query As ADODB.Command
        Set query = New ADODB.Command
        'Build the command to pass to the data source.
        With query
            .ActiveConnection = database
            .CommandText = "SELECT DISTINCT Item FROM Table"
            .CommandType = adCmdText
        End With
        Dim records As ADODB.Recordset
        'Execute the command to retrieve the recordset.
        Set records = query.Execute()

        Do While Not records.EOF
            Debug.Print records("Item")
            records.MoveNext
        Loop
    End If
CleanExit:
    If Not records Is Nothing Then records.Close
    If Not database Is Nothing And database.State = adStateOpen Then
        database.Close
    End If
    Exit Sub
Handler:
    Debug.Print "Error " & Err.Number & ": " & Err.Description
    Resume CleanExit
End Sub
```

Note that commands sent to the data source are **vulnerable to SQL injection**, either intentional or unintentional. In general, queries should not be created by concatenating user input of any kind. Instead, they should be parameterized (see Creating parameterized commands).

Section 37.4: Executing non-scalar functions

ADO connections can be used to perform pretty much any database function that the provider supports via SQL. In this case it isn't always necessary to use the `Recordset` returned by the `Execute` function, although it can be useful for obtaining key assignments after INSERT statements with @@Identity or similar SQL commands. Note that the example below uses the `OpenDatabaseConnection` function from the Making a connection to a data source example for the purpose of brevity.

```
Public Sub UpdateTheFoos()
    On Error GoTo Handler
    Dim database As ADODB.Connection
    Set database = OpenDatabaseConnection(SomeDSN)

    If Not database Is Nothing Then
        Dim update As ADODB.Command
        Set update = New ADODB.Command
        'Build the command to pass to the data source.
        With update
            .ActiveConnection = database
            .CommandText = "UPDATE Table SET Foo = 42 WHERE Bar IS NULL"
            .CommandType = adCmdText
            .Execute        'We don't need the return from the DB, so ignore it.
        End With
    End If
CleanExit:
    If Not database Is Nothing And database.State = adStateOpen Then
        database.Close
    End If
    Exit Sub
Handler:
    Debug.Print "Error " & Err.Number & ": " & Err.Description
    Resume CleanExit
End Sub
```

Note that commands sent to the data source are **vulnerable to SQL injection**, either intentional or unintentional. In general, SQL statements should not be created by concatenating user input of any kind. Instead, they should be parameterized (see Creating parameterized commands).