Connecting to an Excel Workbook with ADO

Using the Microsoft Jet provider ADO can connect to an Excel workbook. Data can be read from the workbook and written to it although, unlike writing data to multi-user databases, you need to be aware that conflicts might arise if you try to write data to a workbook that may currently be open with another user.

The Connection String for Excel

When making a connection to an Excel workbook, supply the path to the file as the Data Source as if you were connecting to a database file. You then need to specify which driver to apply using the *Extended Properties* parameter which also allows you to specify whether or not the first row of data contains headings (the default is *Yes*). This code listing connects to an Excel workbook with the filename *fsADOConnectExcel* located in the root of the C: drive...

```
Sub ConnectToExcel()
    Dim cnn As New ADODB.Connection
    Dim rst As New ADODB.Recordset
    cnn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
             "Data Source=C:\fsADOConnectExcel.xls;" & _
             "Extended Properties=""Excel 8.0;HDR=Yes"";"
```

The *Extended Properties* parameter should refer to the appropriate version of Excel:

Excel Version	Extended Properties String
Excel 97	Excel 97;
Excel 2000/2002/2003	Excel 8.0;

Having made the connection to the workbook file it is now necessary to locate the data to be opened as a recordset. When opening your Excel recordset you can choose to refer to a *Worksheet Name*, a *Range Address* or a *Named Range* in a workbook. If your data has a header row ADO uses the headings as field names. Here are some examples:

Using the Worksheet Name

Here a recordset is opened using the data on a worksheet named *Staff List*. Note that when specifying the name of the worksheet you must add a dollar sign (**$**) to the end of the name…

```
Sub ConnectToExcelUsingSheetName ()
    Dim cnn As New ADODB.Connection
    Dim rst As New ADODB.Recordset
    cnn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
             "Data Source=C:\fsADOConnectExcel.xls;" & _
             "Extended Properties=""Excel 8.0;HDR=Yes"";"
    rst.Open "SELECT * FROM [Staff List$];", cnn, adOpenStatic, adLockReadOnly
```

If you specify only the worksheet name then ADO will look along the columns until it finds one containing data, then down the rows until it finds one containing data. It will then use everything it finds as the recordset.

If you have more than one block of data on the worksheet then ADO may return unexpected results. In this case the safest way to get the correct data is to be more specific and use either named ranges or range addresses.

Using a Named Range

If the Excel data occupies a named range then that name can be used to define the recordset. In this example a recordset is opened using a range of cells named *SalesData*…

```
rst.Open "SELECT * FROM [SalesData];", cnn, adOpenStatic, adLockReadOnly
```

Since any range name can be used only once in a workbook there is no need to specify a worksheet name when you use range names.

Using a Range Address

If you know the exact address of the range of cells containing your Excel data you can use that to define the recordset…

```
rst.Open "SELECT * FROM [A1:D10];", cnn, adOpenStatic, adLockReadOnly
```

If you do not specify the sheet name ADO will assume that the range referred to is on the first worksheet. If it is on a different worksheet you must also specify the worksheet name. Remember to add a dollar sign after the worksheet name between it and the range address.

```
rst.Open "SELECT * FROM [Sales Data$A1:F50];", cnn, adOpenStatic, adLockReadOnly
```

Alternative Syntax Using the Execute Method

Instead of using an SQL statement to define the recordset you can use the **Execute** method of the connection object to return a recordset. In this example a recordset is returned from the *Staff List* worksheet:

```
Sub ConnectToExcelUsingConnectionExecute()
    Dim cnn As New ADODB.Connection
    Dim rst As New ADODB.Recordset
    cnn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
             "Data Source=C:\fsADOConnectExcel.xls;" & _
             "Extended Properties=""Excel 8.0;HDR=Yes"";"
    Set rst = cnn.Execute("[Staff List$]")
```

Note the use of the **Set** keyword when assigning the recordset to the object variable, and the quote marks outside the square brackets around the worksheet name.

The same syntax can be applied when reading data from a range address or a named range e.g.:

```
Set rst = cnn.Execute("[Staff List$A1:D50]")    ' a range address

Set rst = cnn.Execute("[StaffData]")    ' a named range
```

Unlike the recordset **Open** method this method alone does not allow you to specify the additional parameters such as *CursorType* and *LockType* so if you want anything other than the default settings be sure to set these recordset properties *before* the *Execute* statement:

```
Sub ConnectToExcelUsingConnectionExecute()
    Dim cnn As New ADODB.Connection
    Dim rst As New ADODB.Recordset
    cnn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
             "Data Source=C:\fsADOConnectExcel.xls;" & _
             "Extended Properties=""Excel 8.0;HDR=Yes"";"
    With rst
        .CursorType = adOpenStatic
        .LockType = adLockReadOnly
    End With
    Set rst = cnn.Execute("[Staff List$]")
```

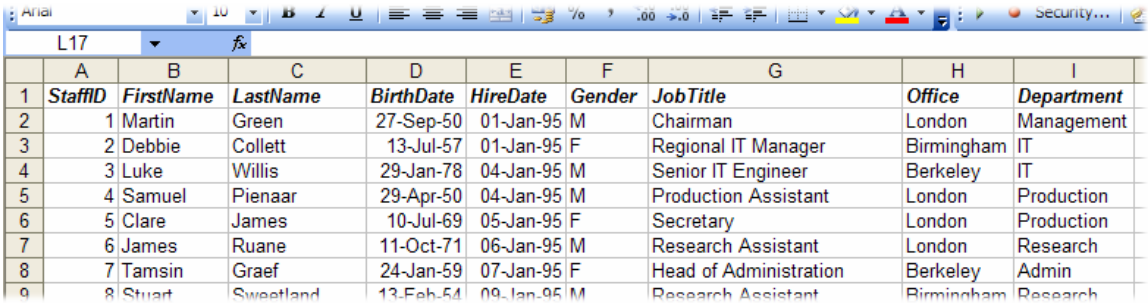
Working with the Recordset

An Excel recordset opened with ADO is read-only so changes and additions cannot be made to the source data. Operations must be concerned with reading the data. Whilst this does avoid complications that might arise due to sharing an error could still occur if another user is currently editing a cell in the recordset's range. To avoid this modify the workbook to allow shared editing Open the workbook in Excel and choose **Tools > Share Workbook** and place a tick in the checkbox marked *Allow changes by more than one user at the same time.* Then save the workbook to enable shared editing.

The above code listings are incomplete, showing just that section of code required to open a recordset. The following examples are complete procedures including the necessary error handlers and exit routines which should accompany all such work.

Looping Through a Recordset

This routine iterates a record at a time through a recordset derived from an Excel worksheet and writes selected information to the Immediate Window of the Visual Basic Editor. Here's how the data looks on the Excel worksheet (*Fig. 1*):



	A	B	C	D	E	F	G	H	I
1	*StaffID*	*FirstName*	*LastName*	*BirthDate*	*HireDate*	*Gender*	*JobTitle*	*Office*	*Department*
2	1	Martin	Green	27-Sep-50	01-Jan-95	M	Chairman	London	Management
3	2	Debbie	Collett	13-Jul-57	01-Jan-95	F	Regional IT Manager	Birmingham	IT
4	3	Luke	Willis	29-Jan-78	04-Jan-95	M	Senior IT Engineer	Berkeley	IT
5	4	Samuel	Pienaar	29-Apr-50	04-Jan-95	M	Production Assistant	London	Production
6	5	Clare	James	10-Jul-69	05-Jan-95	F	Secretary	London	Production
7	6	James	Ruane	11-Oct-71	06-Jan-95	M	Research Assistant	London	Research
8	7	Tamsin	Graef	24-Jan-59	07-Jan-95	F	Head of Administration	Berkeley	Admin

Fig. 1 Data on an Excel worksheet.

Here is the code:

```
Sub ConnectToExcelUsingSheetName()
    On Error GoTo ConnectToExcelUsingSheetName_Err
' Declare variables
    Dim cnn As New ADODB.Connection
    Dim rst As New ADODB.Recordset
' Open the connection then open the recordset
    cnn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
             "Data Source=C:\fsADOConnectExcel.xls;" & _
             "Extended Properties=""Excel 8.0;HDR=Yes"";"
    rst.Open "SELECT * FROM [Staff List$];", cnn, adOpenStatic, adLockReadOnly
' Loop through the recordset and send data to the Immediate Window
    rst.MoveFirst
    Do
        Debug.Print rst![FirstName] & " " & rst![LastName] & " (" & rst![JobTitle] & ")"
        rst.MoveNext
    Loop Until rst.EOF
' Tidy up
ConnectToExcelUsingSheetName_Exit:
    On Error Resume Next
    rst.Close
    cnn.Close
    Set rst = Nothing
    Set cnn = Nothing
    Exit Sub
ConnectToExcelUsingSheetName_Err:
    MsgBox Err.Number & vbCrLf & Err.Description, vbCritical, "Error!"
    Resume ConnectToExcelUsingSheetName_Exit
End Sub
```

After making the connection to the Excel workbook (named *fsADOConnectExcel.xls* and located in the root of the C:\ drive) the procedure opens a recordset based a specified worksheet (named *Staff List*). The code moves to the first record then loops through the records pausing at each to send a concatenated string (*FirstName*, space, *LastName*, space and left bracket, *JobTitle*, right bracket) to the Immediate Window using the *Debug.Print* statement. The result looks like this (*Fig. 2*):

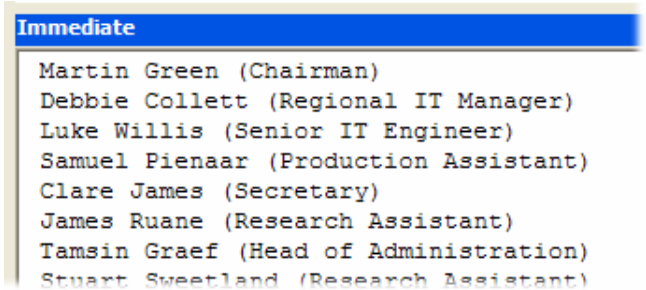


Fig. 2 Data from the recordset is sent to the Immediate Window.

Using a Variant Array

An array is a kind of variable that can hold many separate pieces of data and is an ideal vehicle for holding a recordset. For this procedure the array is declared as a variant to allow it to hold a variety of different data types. In this example the recordset is written into the array then read from the array into a new table in the database:

```
Sub ReadFromExcelUsingArray()
    On Error GoTo ReadFromExcelUsingArray_Err
' Declare variables
    Dim cnn1 As New ADODB.Connection
    Dim cnn2 As New ADODB.Connection
    Dim rst1 As New ADODB.Recordset
    Dim rst2 As New ADODB.Recordset
    Dim i As Integer
    Dim x As Integer
    Dim strSQL As String
    Dim arrData() As Variant
' Open the connection to Excel then open the recordset
    cnn1.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
              "Data Source=C:\fsADOConnectExcel.xls;" & _
              "Extended Properties=""Excel 8.0;HDR=Yes"";"
    rst1.Open "SELECT * FROM [Staff List$];", cnn1, adOpenStatic, adLockReadOnly
' Build a new table in Access using SQL
    strSQL = "CREATE TABLE fsADOConnectExcel ("
    For i = 0 To rst1.Fields.Count - 1
        strSQL = strSQL & "[" & rst1.Fields(i).Name & "] TEXT(50),"
    Next i
    strSQL = Left(strSQL, Len(strSQL) - 1)
    strSQL = strSQL & ");"
    DoCmd.RunSQL strSQL
' Read the Excel recordset into a variant array
    arrData = rst1.GetRows(rst1.RecordCount)
' Open the new table as a recordset
    Set cnn2 = CurrentProject.Connection
    rst2.Open "SELECT * FROM fsADOConnectExcel", cnn2, adOpenDynamic, adLockOptimistic
' Write records from variant array into table
    For i = 0 To rst1.RecordCount - 1
        With rst2
            .AddNew
            For x = 0 To rst1.Fields.Count - 1
                .Fields(x).Value = arrData(x, i)
            Next x
            .Update
        End With
    Next i
' Tidy up
    Application.RefreshDatabaseWindow
    MsgBox "Finished importing data from Excel"
ReadFromExcelUsingArray_Exit:
    On Error Resume Next
    rst1.Close
    rst2.Close
    cnn1.Close
    cnn2.Close
    Set rst1 = Nothing
    Set rst2 = Nothing
    Set cnn1 = Nothing
    Set cnn2 = Nothing
    Exit Sub
ReadFromExcelUsingArray_Err:
    MsgBox Err.Number & vbCrLf & Err.Description, vbCritical, "Error!"
    Resume ReadFromExcelUsingArray_Exit
End Sub
```

The procedure assumes that you do not know the field names nor how many rows of data there are. Note that two connections and two recordsets are declared, the first being the Excel source and the second the Access destination. The connection to Excel is opened and the recordset created in the usual way.

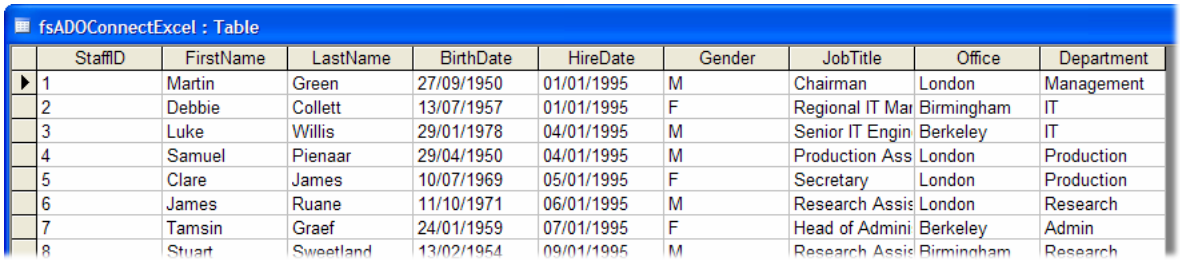
With the recordset in memory the procedure reads the names of the fields and builds an SQL *CREATE TABLE* statement (not knowing the data types in advance it makes all the fields 50 character Text fields). It then runs the SQL statement to create a new table in Access then reads the recordset into the variant array using the *GetRows* statement. This automatically resizes the array to fit the required number of rows and columns (I have specified the number of records to return using the recordset's *RecordCount* property but could have omitted this since if this parameter is not supplied all rows are returned anyway).

A second connection is made, this time to the current database, and the newly created table opened as a recordset. Note that whilst the Excel source data was opened as a read-only recordset, the destination recordset is opened in dynamic form so that records can be added.

A pair of nested loops are used to write the data from the array into the table from the variant array. Using variant arrays in this way is very fast!

After refreshing the database window to ensure that the new table is shown, and notifying the user that the process is complete, the recordsets and connections are closed and their variables purged from memory so that the procedure can safely terminate.

Here's the resulting Access table (*Fig. 3*):



fsADOConnectExcel : Table

StaffID	FirstName	LastName	BirthDate	HireDate	Gender	JobTitle	Office	Department
1	Martin	Green	27/09/1950	01/01/1995	M	Chairman	London	Management
2	Debbie	Collett	13/07/1957	01/01/1995	F	Regional IT Mar	Birmingham	IT
3	Luke	Willis	29/01/1978	04/01/1995	M	Senior IT Engin	Berkeley	IT
4	Samuel	Pienaar	29/04/1950	04/01/1995	M	Production Ass	London	Production
5	Clare	James	10/07/1969	05/01/1995	F	Secretary	London	Production
6	James	Ruane	11/10/1971	06/01/1995	M	Research Assis	London	Research
7	Tamsin	Graef	24/01/1959	07/01/1995	F	Head of Admini	Berkeley	Admin
8	Stuart	Sweetland	13/02/1954	09/01/1995	M	Research Assis	Birmingham	Research

Fig. 3 An new Access table created from data in Excel.