ABSTRACT

The Microsoft Office suite is ubiquitous in most large organisations these days. And regardless of how and where the enterprise intelligence computing goes on, the suite of Office products needs to be able to integrate with this enterprise intelligence. With the release of SAS Integration Technologies the unrivalled analytical and business intelligence capabilities of SAS can now be delivered to the desktop using the Microsoft Office suite. This paper will introduce some components of SAS Integration Technologies, and then show how to use these components to integrate SAS into Microsoft Office Products without the need for SAS on the user's computer. This paper is geared for any SAS programmer who has a good grasp of VBA.

There are few SAS programmers and/or analysts who have not heard the refrain ‘Can I have that in an Excel spreadsheet?’ And, once provided, the additional and inevitable refrain ‘Can you run it again with this one change?’. If you have SAS/Access for PC File Formats, generating the spreadsheets is not too big an issue, assuming you have time to make the change and run the programme. And if you do not have SAS/Access for PC File formats, then you have yet another layer of conversion, additional time, and of course the window of opportunity for errors to creep in. And if the results are required in a MS Word compatible format or a MS Access compatible format there are yet other problems you may face. Many people turned to Dynamic Data Exchange (DDE) to try to alleviate these problems. In this paper I will introduce a more robust set of tools aimed at sharing data between SAS and non-SAS applications – SAS Integration Technologies.

I AM IN THE OFFICE BUT SAS IS OUT THERE

SAS is running out on the network (say on a Windows 2000 server) and you have to provide a user with a table to be included into an Excel spreadsheet. And, SAS in not installed on the user’s computer. What to do? One solution is run SAS on the remote computer, create the output, copy it to a location accessible to the user, then notify the user it is ready. The final output generation is simple thanks to ODS (Listing 1); Excel will gladly accept the CSV file created through ODS. Although Word does not like this format, you can appease your Word users with one simple change (Listing 2). Word will gladly accept the RTF file created through ODS. And if you do not know whether it will be Word or Excel that will use the data, you can always create an HTML output (Listing 3). Finally, MS Access and Powerpoint can also accept HTML output. So life is not that bad. Until you get asked for changes. Never happens, right.

SAS IS IN THE OFFICE WITH ME

New in SAS v8, Integration Technologies is a set of tools that allow you to access the power of the SAS system from a variety of programming environments – Java, Visual Basic etc.. In this paper, we will be looking at using some Windows desktop tools and components to access SAS. In most of the examples we will be using Excel VBA as the client programming language, Microsoft’s Active Data Objects (ADO) as the data access component, and the SAS Integrated Object Model (IOM) to open SAS to the Excel client. Some of the examples will show how to perform the same task from different MS Office products. Finally we will look deeper into the MS Office environment and how to put links to SAS right onto the client (e.g. Excel) menu.

First, let us briefly examine the SAS Integrated Object Model (for a complete description of SAS Integration Technologies, and the IOM, refer to the SAS web site support.sas.com/rnd/itech).

The root of the IOM hierarchy is the SAS Workspace object; when instantiated by the Workspace Manager within a client programme, the SAS Workspace object can be thought of as a SAS session. Virtually all of the functionality you would have in a batch SAS session is available to you through the workspace object. The Workspace Manager creates the SAS Workspace objects on an IOM Server. In the Windows environment there are three ways the Workspace Manager can create a SAS Workspace:

Through local COM if the SAS Server runs on the same machine as the client
Through DCOM if the SAS Server runs on another machine that supports DCOM

Figure 1.
IOM Hierarchy
Through the IOM Bridge for COM if the SAS Server runs on another machine that does not support COM/DCOM functionality (Unix/OS390)

Regardless of how the SAS Workspace is created (COM, DCOM, IOM Bridge), it still offers the same set of services – DataService, FileService, LanguageService, and Utilities.

The LanguageService component provides methods to submit SAS code to the IOM Server as well as retrieve log and list outputs. If you take advantage of the SAS Output Delivery System (ODS) you can use the ResultsPackageService to retrieve collected items. While the programme is executing, the LanguageService raises events (e.g. step begin, step end), which will allow you to monitor the progress.

The DataService and FileService provide methods to access SAS libraries through librefs or host system files through filerefs. The full range of library and file manipulation tools is available through these services. Microsoft's ADO/OLE DB data model is used to share data between the client application and the SAS IOM server. The ActiveX Data Object (ADO) model is shown in Figure 2.

MICROSOFT ADO

In order to share data between SAS and Excel we will need to understand a bit about ADO. For the purposes of this paper there are two objects of interest – the Connection object and the Recordset object.

The Connection object provides properties to define the source of the data, and methods to manage the link between the client to the datasource. The Recordset object uses the Connection object to return data to the client. The Fields collection of the Recordset object provides data about the contents of the recordset.

DO YOU HAVE ALL OF YOUR TOOLS?

In order to follow the examples in this paper, you will have to have SAS v8.x installed; as part of the installation procedure be sure the Integration Technologies components are installed. Microsoft Data Access Components (MDAC) v2.1 or higher should also be installed; normally this will be installed along with SAS Integration Technologies. The examples that follow were developed under Windows 2000 Professional sp2 using SAS v8.2. The office products are all from Office 2000.

In order to use the SAS IOM within Excel we will need to make sure that Excel has the proper references to the objects. To do this, start Excel and follow these steps:

1. From an empty spreadsheet open the Visual Basic Editor (Tools…Macro…Visual Basic Editor)
2. create a new module (Insert…Module)
3. add the references to the SAS IOM and the SAS Workspace manager (Tools…References – in the dialogue box, scroll down and select the SAS objects)
4. add the references to the Microsoft ActiveX Data Objects (Tools…References – in the dialogue box scroll down to Microsoft ActiveX Data Objects and select the highest version available)

A similar process will need to be followed for each of the other Office products.

FIRST DAY AT THE OFFICE

Let's start with a simple connection and retrieval of data; in the first case we will retrieve the contents of the table *sasuser.shoes*. To keep the demonstration as simple as possible and to highlight the IOM components the Excel worksheet will not use common spreadsheet components such as forms and buttons. The first example will simply return the contents of *sasuser.shoes* and display the contents in the Excel immediate window. Let us examine each of the lines of code here (see Listing 1). NOTE: watch carefully for code that spans multiple lines. Although I have tried to catch all such spans and add the 'line to be
continued’ character _ (the underscore), the transfer from Excel to the word processor may have led to word wrap, and consequently code that will not compile.

First, whenever using VBA modules, always set Option Explicit. This option requires you to declare all variables used. If this option is not set errors in the form of misspelled variables can easily creep into your programme. The next three declarations:

Dim swsSAS As SAS.Workspace
Dim rsSAS As New ADODB.Recordset
Dim swmWM As New _
SASWorkspaceManager.WorkspaceManager

declare these objects to be global to the module. The use of the New keyword indicates that the objects should be created when the programme starts. These objects were declared global to the module so they could be available to any function or subroutine within the module. The subroutine Test does all of the work in this module.

The command:

Set swsSAS =
swmWM.Workspaces.CreateWorkspaceByServer delimiter _
VisibilityProcess, Nothing, delimiter, delimiter, xmlInfo)

creates a SAS workspace (swsSAS) on the local machine. The third parameter (the VBA keyword Nothing) indicates the SAS server is on the local machine. If the SAS server were to be located on another machine then an appropriate server definition would have to be passed

The command (which should be on one line):

cnnIOM.Open "Provider=sas.iomprovider.1; SAS
Workspace ID=" & swsSAS.UniqueIdentifier

opens the data connection between the SAS IOM server and the Excel client. The “Provider=sas.iomprovider.1” option indicates the particular SAS dataprovider we will be using (there are three available providers). The “SAS Workspace ID= & swsSAS.Uniqueidentifier” option tells the connection which workspace it is dealing with.

The command:

rsSAS.Open "sashelp.shoes", cnnIOM, adOpenDynamic, delimiter_
adLockPessimistic, ADODB.adCmdTableDirect

opens the SAS table sashelp.shoes for update. The connection object, cnnIOM, identifies where the data reside (a local SAS workspace), the adOpenDynamic keyword indicates the data are updateable.

A recordset can be positioned in one of three locations, before the first record or the beginning of the file (BOF), after the last record or the end of the file (EOF), or on an active record; if there are no records in the recordset you cannot move to an active record. A common way to test for an empty dataset is to see if both the BOF and EOF properties are true; if both properties are true, there are no records in the recordset. The statement

If Not (rsSAS.BOF And rsSAS.EOF) Then

then checks if there are any records in the recordset, proceeding only if there are records. There are a number of methods used to navigate a recordset; some of the common ones are:

MoveFirst - move to the first record
MoveLast - move to the last record
MoveNext – move to the next record
MovePrevious – move to the previous record

Within the conditional If statement, we loop through all of the records in the recordset, listing the contents in the VBA immediate window using the Debug.Print method.

rsSAS.MoveFirst
Do While Not rsSAS.EOF
  Debug.Print rsSAS!region, rsSAS!product,_
  rsSAS!subsidiary, rsSAS!stores, rsSAS!sales,_
  rsSAS!inventory, rsSAS!returns
rsSAS.MoveNext
Loop

After looping through all of the records, we close all of the objects we had opened and explicitly destroy them by setting them to Nothing. If objects are not properly closed and destroyed, the memory they occupied is not freed (memory leak) often resulting in your programme slowing drastically and possibly crashing altogether.

WHAT FIELD ARE YOU IN?

The example in Listing 1 allowed us to display the contents of the table sasuser.shoes; however, we had to explicitly identify all of the fields in the recordset. In the next example (Listing 2) we will see how to identify the fields in each record. The following code is the segment which lists the fields:

For Each fld In rsSAS.Fields
  Debug.Print fld.Name, fld.Type
Next

If you recall from the ADO hierarchy model (Figure 2) the recordset object has a Fields collection; this code is simply stepping through all of the fields in the collection. The For Each …. Next construct is a common method to iterate over a collection.

LETS BE OBJECTIVE.

How do you find the options and properties for these objects we are using? If manuals were regularly distributed with software, you could read the manual. Now manuals are replaced with on-line help. Unfortunately, navigating the help files is not always easy or productive. Fortunately there is a way to get the methods and properties of the objects – the VBA object browser. Within the VBA editor select View… Object Browser; the shortcut key id F2 (Figure 3).
The example in Figure 3 shows the members (properties and methods) of the recordset object.

LET'S EXCELERATE THE PROCESS

Now that we have been able to start SAS from our Excel spreadsheet, let's actually populate a worksheet (Listing 3). This example builds upon the previous one. We will iterate over the Fields collection to put out column headers (fld.Name). In addition we will check for character fields (fld.Type = adWChar) and set the column widths to be 2 characters wider than the actual (defined) width (fld.DefinedSize + 2). We will also bold the titles and set the cell border to a bottom underline. After displaying the column headers, we then process every record in the recordset as before; within each record we iterate over the Fields collection and populate the cells with the field value (fld.Value). After populating the worksheet, we move to the cell A2.

SUBMIT

Ok, we can read sasuser.shoes. My boss will really find those data useful!! Ok, maybe he will, maybe she won’t. But a DATA Step, that we know would be useful. In the next example (Listing 4) we use the LanguageService to submit a simple data step and return the results. As with the first example, the output will be displayed in the VBA Immediate window. The new piece to the puzzle is the line (the quoted part should all be on one line):

swsSAS.LanguageService.Submit "data a; do customer=1 to 0;quantity=customer*customer;pizza='Pepperoni';output;end;run;"

This will create a dataset with 10 records and 3 variables. And now with the ability to submit SAS code we can add some real v8 power.

HERE'S WHERE WE KEEP THE DATA.

In order to access some real data we need to use the SAS Workspace DataService to assign a libref to an existing SAS Library (Listing 5). In this example we assign the libref CARD to the directory D:\Cardiac using the command

Set libref = swsSAS.DataService.AssignLibref("card", "", "d:\cardiac", "")

Then, in the SAS code in the submit command, we can reference datasets in the specific libref CARD.

BEFORE YOU GO HOME

These examples have been kept simple to highlight a few of the aspects of SAS Integration Technologies. By no means are they exhaustive of the power of SAS Integration Technologies. However, they should start you on your way. To see how to link SAS and Integration Technologies into Excel and Access menu items see the paper by Darren Key and David Shamlin from SUGI 27.

To get started you should have SAS with Integration Technologies installed on your desktop. Once you have the programmes working using a local SAS Server it is a matter of changing only a few options and you are ready to run the same programmes against remote SAS servers. To deploy your applications you need only the SAS Client components and the client application (e.g. Excel) on the desktop and SAS Integration Technologies components on the remote SAS Server.

SAS, SAS Integration Technologies, and SAS Alliance Partner are registered trademarks of SAS Institute Inc. in the USA and other countries
Other brand and product names are registered trademarks or trademarks of their respective companies.

REFERENCES

For a complete overview of SAS Integration Technologies see support.sas.com/rnd/itech

Wrox Press, Birmingham

SAMS, Indianapolis IN

Key, Darren and David Shamlin “Using SAS® Data To Drive Microsoft Office”
Proceedings of the Twenty-Seventh Annual SAS Users Group International Conference.
About the Author

Peter is SAS Certified Professional V8, SAS Certified Professional V6, and SAS Certified Professional - Data Management V6. In addition his company, Fernwood Consulting Group Inc. is a SAS Alliance Partner.

If you have any questions or comments you can contact Peter at: Fernwood Consulting Group Inc., 288 Laird Dr., Toronto ON M4G 3X5 Canada
Voice: (416)429-5705
E-mail: peter@fernwood.ca

LISTINGS

Listing 1

Option Explicit ' always set option explicit
Dim swsSAS As SAS.Workspace
Dim rsSAS As New ADODB.Recordset
Dim swmWM As New SASWorkspaceManager.WorkspaceManager

Public Sub test()
Dim cnnIOM As New ADODB.Connection
Dim xmlInfo As String

' Create a local SAS workspace.
Set sswSAS = swmWM.Workspaces.CreateWorkspaceByServer("", VisibilityProcess, Nothing, ",", ",", xmlInfo)

' Open a connection to the workspace
  cnnIOM.Open "Provider=sas.iomprovider.1; SAS Workspace ID=" & sswSAS.UniqueIdentifier

' Associate the Recordset object with the SAS data set.
rsSAS.Open "sashelp.shoes", cnnIOM, adOpenDynamic, adLockPessimistic, ADODB.adCmdTableDirect
For Each fld In rsSAS.Fields
    Debug.Print fld.Name, fld.Type
Next
rsSAS.Close
Set rsSAS = Nothing
  cnnIOM.Close
Set cnnIOM = Nothing
SwmWM.Workspaces.RemoveWorkspaceByUUID swsSAS.UniqueIdentifier
SwsSAS.Close
Set swsSAS = Nothing
Set swmWM = Nothing
End Sub

Listing 2

Option Explicit ' always set option explicit
Dim swsSAS As SAS.Workspace
Dim rsSAS As New ADODB.Recordset
Dim swmWM As New SASWorkspaceManager.WorkspaceManager

Public Sub test()
Dim cnnIOM As New ADODB.Connection
Dim xmlInfo As String
Dim fld As Field

' Create a local SAS workspace.
Set sswSAS = swmWM.Workspaces.CreateWorkspaceByServer("", VisibilityProcess, Nothing, ",", ",", xmlInfo)

' Open a connection to the workspace
  cnnIOM.Open "Provider=sas.iomprovider.1; SAS Workspace ID=" & sswSAS.UniqueIdentifier

' Associate the Recordset object with the SAS data set.
rsSAS.Open "sashelp.shoes", cnnIOM, adOpenDynamic, adLockPessimistic, ADODB.adCmdTableDirect
For Each fld In rsSAS.Fields
    Debug.Print fld.Name, fld.Type
Next
rsSAS.Close
Set rsSAS = Nothing
  cnnIOM.Close
Set cnnIOM = Nothing
SwmWM.Workspaces.RemoveWorkspaceByUUID swsSAS.UniqueIdentifier
SwsSAS.Close
Set swsSAS = Nothing
Set swmWM = Nothing
End Sub

Listing 3

Option Explicit ' always set option explicit
Dim swsSAS As SAS.Workspace
Dim rsSAS As New ADODB.Recordset
Dim swmWM As New SASWorkspaceManager.WorkspaceManager

Public Sub test()
Dim cnnIOM As New ADODB.Connection
Dim xmlInfo As String
Dim count As Integer
Dim fld As Field
Dim row As Long

' Create a local SAS workspace.
Set sswSAS = swmWM.Workspaces.CreateWorkspaceByServer("", VisibilityProcess, Nothing, ",", ",", xmlInfo)

' Open a connection to the workspace
Listing 4

Option Explicit ' always set option explicit
Dim swsSAS As SAS.Workspace
Dim rsSAS As New ADODB.Recordset
Dim cnnIOM As New ADODB.Connection
Dim xmlInfo As String
Dim swmWM As New SASWorkspaceManager.WorkspaceManager

Public Sub test()
    Dim cnnIOM As New ADODB.Connection
    Dim xmlInfo As String

    ' Associate the Recordset object with the SAS data set.
    rsSAS.Open "sashelp.shoes", cnnIOM, adOpenDynamic, adLockPessimistic, ADODB.adCmdTableDirect
    ' SELECT the first sheet and freeze the panes on the 2nd line
    Worksheets("sheet1").Activate
    Range("A2").Select
    ActiveWindow.FreezePanes = True
    If Not (rsSAS.BOF And rsSAS.EOF) Then
        Worksheets("sheet1").Activate
        Range("A1").Select
        For Each fld In rsSAS.Fields
            ActiveCell.Value = fld.Name
            ActiveCell.Font.Bold = True
            With ActiveCell.Borders(xlBottom)
                .LineStyle = xlContinuous
                .Weight = xlThin
            End With
        Next
        rsSAS.MoveNext
    If fld.Type = adWChar Then
        Columns(ActiveCell.Column).ColumnWidth = fld.DefinedSize + 2
    Else
        Columns(ActiveCell.Column).ColumnWidth = 12
    End If
    rsSAS.MoveFirst
    Do While Not rsSAS.EOF
        Debug.Print rsSAS!CUSTOMER, rsSAS!QUANTITY, rsSAS!PIZZA, rsSAS!ORDERDATE
        rsSAS.MoveNext
    Loop
   rssSAS.Close
    Set rsSAS = Nothing
    cnnIOM.Close
    Set cnnIOM = Nothing
    swmWM.Workspaces.RemoveWorkspaceByUUID swsSAS.UniqueIdentifier
    swsSAS.Close
    Set swsSAS = Nothing
    Set swmWM = Nothing
End Sub

Listing 5

Option Explicit ' always set option explicit
Dim swsSAS As SAS.Workspace
Dim rsSAS As New ADODB.Recordset
Dim cnnIOM As New ADODB.Connection
Dim xmlInfo As String
Dim count As Long
Dim libref As SAS.libref
Dim fld As Field

Public Sub test()
    Dim cnnIOM As New ADODB.Connection
    Dim xmlInfo As String

    ' Create a local SAS workspace.
    Set swsSAS = swmWM.Workspaces.CreateWorkspaceByServer("", VisibilityProcess, Nothing, '', '', xmlInfo)
    ' Use LanguageService
    swsSAS.LanguageService.Submit "data a; do customer=1 to 10;quantity=customer*customer;
    pizza='Pepperoni';output;end;run;"
    ' Open a connection to the workspace
    cnnIOM.Open "Provider=sas.iomprovider.1; SAS Workspace ID=" & swsSAS.UniqueIdentifier
    ' Associate the Recordset object with the SAS data set.
    rsSAS.Open "work.a", cnnIOM, adOpenDynamic, adLockPessimistic, ADODB.adCmdTableDirect
    If Not (rsSAS.BOF And rsSAS.EOF) Then
        rsSAS.MoveFirst
        Do While Not rsSAS.EOF
            Debug.Print rsSAS!CUSTOMER, rsSAS!QUANTITY, rsSAS!PIZZA, rsSAS!ORDERDATE
            rsSAS.MoveNext
        Loop
    End If
    rssSAS.Close
    Set rsSAS = Nothing
    cnnIOM.Close
    Set cnnIOM = Nothing
    swmWM.Workspaces.RemoveWorkspaceByUUID swsSAS.UniqueIdentifier
    swsSAS.Close
    Set swsSAS = Nothing
    Set swmWM = Nothing
End Sub
swsSAS.LanguageService.Submit "data a; set
card.revup;run;"

' Open a connection to the workspace
cnnIOM.Open "Provider=sas.iomprovider.1; SAS
Workspace ID=" & swsSAS.UniqueIdentifier

' Associate the Recordset object with the SAS data set.
rsSAS.Open "work.a", cnnIOM, adOpenDynamic,
adLockPessimistic, ADODB.adCmdTableDirect
If Not (rsSAS.BOF And rsSAS.EOF) Then
  For Each fld In rsSAS.Fields
    Debug.Print fld.Name
  Next

  rsSAS.MoveFirst
  Do While Not rsSAS.EOF
    Debug.Print rsSAS!CUSTOMER,
    rsSAS!QUANTITY, rsSAS!PIZZA, rsSAS!ORDERDATE
    rsSAS.MoveNext
  Loop
End If
rsSAS.Close
Set rsSAS = Nothing

swsSAS.DataService.DeassignLibref "card"
swmWM.Workspaces.RemoveWorkspaceByUUID
swsSAS.UniqueIdentifier
swsSAS.Close
Set swsSAS = Nothing
Set swmWM = Nothing
End Sub