Paper SY-13

Bring the Data Warehouse to the Office with SAS® Integration Technologies

Peter Eberhardt, Fernwood Consulting Group Inc., Toronto, ON

ABSTRACT

No matter how sophisticated you're your data warehousing solution may be you will still get the question "Can I have that in EXCEL?" The Microsoft Office suite is ubiquitous in most large organizations these days. And regardless of how and where the enterprise intelligence computing goes on, the suite of Office products needs to be able integrate with this enterprise intelligence, or the time and energy you spend on ensuring data quality can be quickly lost with a sloppy cut and paste into EXCEL or WORD. 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. With SAS v9 came a new product, the SAS Add-in for Microsoft Office. This paper will not address the Add-In, rather it will focus on a programmatic approach to integrating SAS and Excel.

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; but first, lets look at some alternatives to exporting data from SAS to MS Office.

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. If the data you need to export are already available in SAS, say in a dataset or standard report already being produced, it can be as simple as wrapping your code with some ODS statements. For example, if you need to export a dataset to EXCEL, the following code will do this nicely:

```
ODS LISTING CLOSE;
ODS CSV FILE = 'C:\temp\shoes.csv';
PROC PRINT DATA=sashelp.shoes;
Run;
ODS CSV CLOSE;
ODS LISTING;
```

CSV (Comma Separated Values) is a compact style that EXCEL can import directly; it is useful if you need to export tabular data. If the data you export to EXCEL should also include titles, footnotes, or highly formatted reports, then write to an HTML document, but give the document an XLS extension. This works, since EXCEL can read HTML documents; giving the XLS extension means that the document will be opened by EXCEL rather than your browser of choice. See the paper by Chevell Parker for an excellent overview of using ODS to generate EXCEL output using ODS. Finally, if your export destination will be WORD, then write your output to an RTF (Rich Text Format) file. Although these are useful additions to a developer's toolkit, and in some cases all that may be required to answer the inevitable "Can I have that in EXCEL?" refrain, they are not methods that allow you in integrate MS Office applications into your SAS systems.

INTEGRATION TECHNOLOGIES: SEPARATING THE INTERFACE FROM THE ENGINE

SAS introduced Integration Technologies (IT) in version 8 of the SAS system. It was an introduction that received little fanfare, yet was a significant move by SAS. Integration Technologies moved SAS from a closed, proprietary system

out into the open systems arena; the full analytical power of the SAS system can now be harnessed through your development platform of choice. This means you can develop your front end application using the your corporate development platform, be it Java, C++, VB, the .NET family, even EXCEL yet leverage the backend analytic power of SAS. You can separate your interface from your engine. Moreover, this can be done with have SAS licensed on every desktop upon which the application runs. This paper will focus on the version 8.2 release of Integration Technologies and the Integrated Object Model (IOM) and how to use EXCEL to access SAS through the IOM. Some of the new features available in version 9.1 will be discussed later.

SAS IS IN THE OFFICE WITH ME: THE INTEGRATED OBJECT MODEL

The IOM is the heart of Integration Technologies ability to interface with multiple and diverse programming environments; it is a deceptively simple object model that exposed all of the power of SAS to your programming environment. The following figure shows the IOM hierarchy

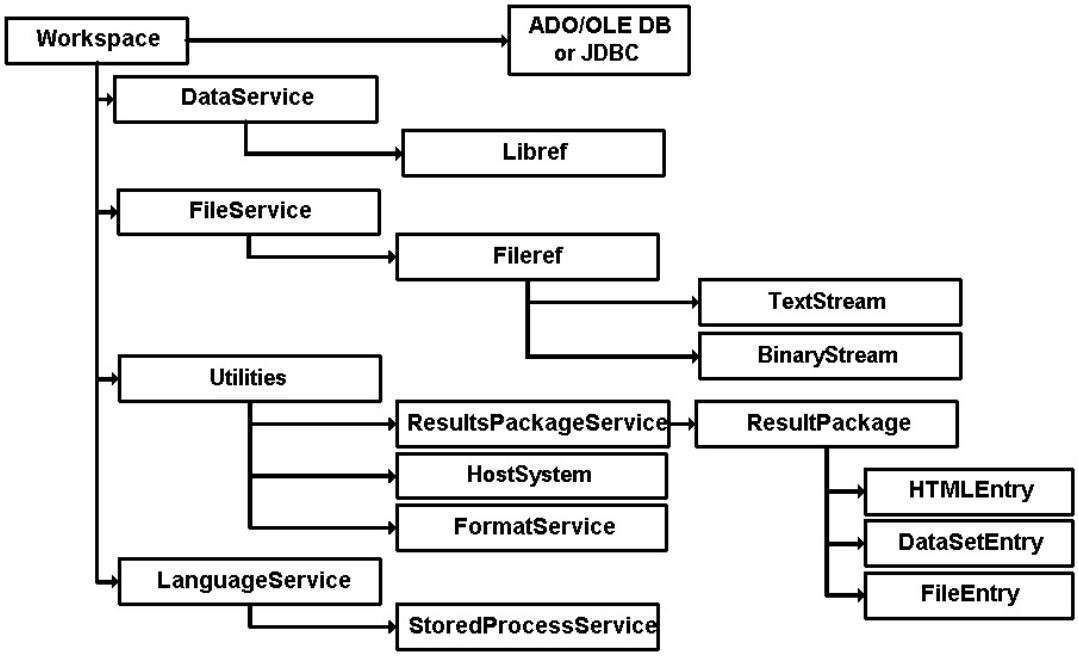

Figure 1
The IOM Hierarchy

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
- 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. In addition to submitting SAS code, the LanguageService allows you to execute SAS Stored Processes – a special format of a SAS programme available on the server. While the programme or submitted code is executing, the LanguageService can be set to raise events (e.g. step begin, step end), which will allow you to monitor the progress.

If you take advantage of the SAS Output Delivery System (ODS) you can use the ResultsPackageService to retrieve collected items.

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 below.

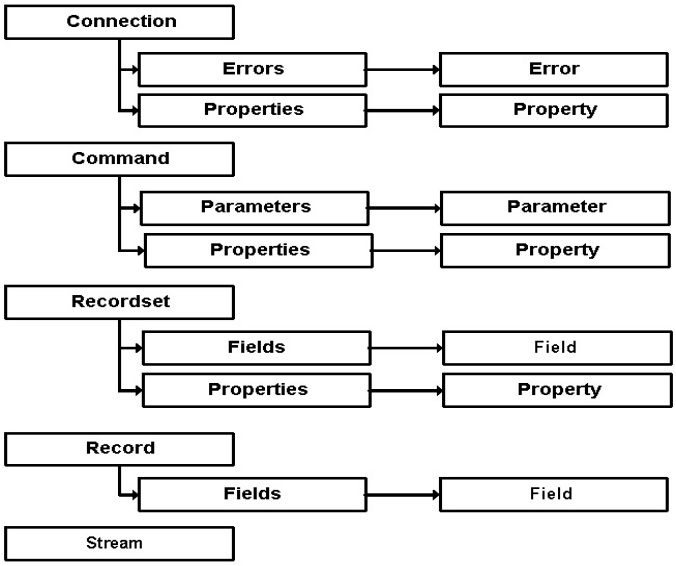

Figure 2
The ADO Hierarchy

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.

REQUIREMENTS

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

From an empty spreadsheet open the Visual Basic Editor (Tools…Macro…Visual Basic Editor)

- Create a new module (Insert…Module)
- 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)
- 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; for a working

spreadsheet that will demonstrate these features contact me by email. 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 ***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("",_
    VisibilityProcess, Nothing, "", "", 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, 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 that 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.

LET'S 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).

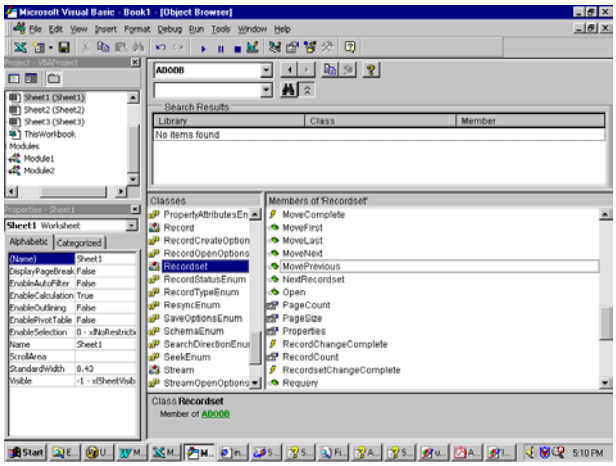

Figure 3
The Object Browser

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 STORE 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. Remember, the programme will be running on the server, so the folder reference (in this example d:\cardiac) must be a folder accessible by the server.

HERE'S WHERE WE STORE THE PROGRAMMES

The real power of Integration Technologies starts to come to the fore when we start using SAS stored processes. A stored process is a SAS programme that can take input parameters; when executed, it can return results to the client. A stored process is normal a text file with normal SAS programme statements (DATA steps, PROC steps etc); the difference between a normal SAS programme file and a SAS stored process is the line:

```
*ProcessBody;
```

All of the statement following this line are standard SAS. However, preceding this line are the input parameters as SAS macro variables. Listing 6 shows a SAS stored process with two input parameters. Note that by assigning values to the SAS macro variable you are explicitly setting a default value; that is, if the client programme does not set the input parameter, the default is used. On the other hand, if the client programme sets a value, then that value is used.

A stored process is executed using a LanguageService command (Listing 7)

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. For an EXCEL spreadsheet which demonstrates using Integration Technologies contact me by email/

VERSION 9.1

NOTE: At the time this paper was written, I did not have version 9.1 of SAS to test the new features. Since some of the functionality of version 9.0 may have changed in version 9.1 so no 9.0 examples are included. Contact me by email for 9.1 examples.

Although the core functionality of the IOM as discussed above is still in place, there are a number of new features in version 9.1. In addition to performance and scalability improvements, there are expanded IOM client services available. Some of the enhancements to the IOM are:

- Takes advantage of the Threaded Kernel technology
- A new Windows Object Manager to create and mange workspace objects
- Access to the SAS Open Metadata Architecture for configuration and security information
- Access to Stored Process Servers to execute and produce output

REFERENCES

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

Green, John (1999) Excel 2000 VBA Programmer's Reference
Wrox Press, Birmingham

Jennings, Roger (1999) Database Developer's Guide with Visual Basic® 6
SAMS, Indianapolis IN

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

Parker, Chevell “Generating Custom Excel Spreadsheets using ODS”
Proceedings of theTwenty-Eighth Annual SAS Users Group International Conference.

CONTACT INFORMATION

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 swsSAS = swmWM.Workspaces.CreateWorkspaceByServer("", VisibilityProcess,
Nothing, "", "", xmlInfo)
' 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 "sashelp.shoes", cnnIOM, adOpenDynamic, adLockPessimistic,_
ADODB.adCmdTableDirect
   If Not (rsSAS.BOF And rsSAS.EOF) Then
       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
   End If
   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 (a=
   Set swsSAS = swmWM.Workspaces.CreateWorkspaceByServer("", VisibilityProcess,_
Nothing, "", "", xmlInfo)
' Open a connection to the workspace (all one line)
cnnIOM.Open "Provider=sas.iomprovider.1; SAS Workspace ID=" &
swsSAS.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 swsSAS = swmWM.Workspaces.CreateWorkspaceByServer("", VisibilityProcess,_
Nothing, "", "", xmlInfo)
' Open a connection to the workspace (all one line)
cnnIOM.Open "Provider=sas.iomprovider.1; SAS Workspace ID=" &
swsSAS.UniqueIdentifier
' 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
         If fld.Type = adWChar Then
```

```
            Columns(ActiveCell.Column).ColumnWidth = fld.DefinedSize + 2
         Else
            Columns(ActiveCell.Column).ColumnWidth = 12
         End If
         col = col + 1
         ActiveCell.Next.Select
      Next
      rsSAS.MoveFirst
      row = 2
      Do While Not rsSAS.EOF
         ActiveSheet.Cells(row, 1).Select
         For Each fld In rsSAS.Fields
            ActiveCell.Value = fld.Value
            ActiveCell.Next.Select
         Next
         row = row + 1
         RsSAS.MoveNext
      Loop
   Range("A2").Select
   End If
   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 4

```
Option Explicit ' always set option explict
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 swsSAS = swmWM.Workspaces.CreateWorkspaceByServer("", VisibilityProcess,
Nothing, "", "", xmlInfo)
    ' Use LanguageService quoted section all on one line
swsSAS.LanguageService.Submit "data a; do customer=1 to
10;quantity=customer*customer; pizza='Pepperoni';output;end;run;"
' Open a connection to the workspace (all one line)
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
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 5

```
Option Explicit ' always set option explict
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 Long
Dim libref  As SAS.libref
Dim fld     As Field
' Create a local SAS workspace.
   Set swsSAS = swmWM.Workspaces.CreateWorkspaceByServer("", VisibilityProcess,_
Nothing, "", "", xmlInfo)
   Set libref = swsSAS.DataService.AssignLibref("card", "", "d:\cardiac", "")
   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
```

LISTING 6

```
%LET LS = 130;
%LET country = ;
*ProcessBody;
OPTIONS linesize = &ls; * use the default if no value passed;
LIBNAME sugi29 'c:\sugi29\data';
DATA sugi29.numbers;
     SET sasuser.customers;
     WHERE country = "&county"; * use the value passed by the client;
RUN;
```

LISTING 7 (SNIPPET)

```
' create the stored process server
   Set sp = sasWS.LanguageService.StoredProcessService
' tell it wher to find the stored processes
   sp.Repository = "file:" & sWorkBookPath & "\procs"
```

```
' and execute the stored process publish.sas with 1 arg
   sp.ExecuteWithResults "publish", "county='CANADA'", rpSAS
```