

Yes! SAS® ExcelXP WILL NOT Create a Microsoft Excel® Graph; But SAS Users Can Command Microsoft Excel to Automatically Create Graphs From SAS ExcelXP Output

William E Benjamin Jr, Owl Computer Consultancy, LLC, Phoenix, AZ

ABSTRACT

With everything becoming more “Security Aware”, hearing that Microsoft Excel 2007 can ignore DDE is no surprise. The SAS® ODS Tagset ExcelXP creates *.xml output, and *.xml output cannot contain graphs. Nowadays, no one wants Excel spreadsheets with any kind of macros in the workbook. Most default security settings will not open “Macro Infected” workbooks (without an authorized Digital Signature – just try to get one of those). So how can SAS Programmers, without “Admin” rights, get graphs into your Excel workbooks? One way is to build them in Excel yourself, one at a time... But, did you really start writing SAS code to build graphs one at a time, “In Excel”? This paper shows you how to create data using SAS, then command Microsoft Excel to read the data, create a graph or fully reformat a worksheet, build a *.xls/*.xlsx file (NOTE – no embedded macros can be saved in *.xlsx files); without putting an Excel macro into the Excel Workbook? And the programs will do it all while you watch, for multiple sheets in a workbook? Over, and Over, and Over again. This paper defines the pieces and parts of SAS code, and Excel code, that you can write, which will allow the creation of a fully integrated system to create and format macro free Excel 2007 workbooks, without downloading any external data or code from the “Forbidden Unsecure Internet”; using SAS Version 9 (SAS Version 8 if internet downloads are available) and Excel 97 and above.

INTRODUCTION

DDE is an old but powerful tool. It can be used to drop data into the middle of an Excel worksheet, or an Excel “Named Range” of cells, without disturbing any of the cells around the rest of the page. This is a great tool for providing data for a graph in Excel, when the exact size of the data is known. The ODS tagset ExcelXP on the other hand writes a *.xml text file. Excel can process these files, but SAS cannot include SAS Graphs in *.xml files. This paper will introduce a simple ‘Hello World’ project that will demonstrate how to use the SAS ODS Tagset “ExcelXP” to create a simple *.xml file and have SAS send it to Excel. Excel will then modify the file and display the results for us to examine. The only action required on the part of the SAS user (after the initial setup of the code) is to run the SAS code. All of the Excel activity is pre-programmed.

A “Hello World” project is a “Proof of Concept” project. When using computers that usually means that if you can write one word to a disk drive and read that word back, then you can fill the space available to you and read it all back. More broadly, if you can show that one thing is possible, then all legal actions are too. This gives the programmer a path from which they can branch out and do anything. The hardest thing is to figure out how to do the first thing. This paper will define a “Hello World” project and extend that to an example routine that will work as a scaffold upon which additional routines can be built.

THE “HELLO WORLD” PROJECT

Well, it was mentioned that there was some setup required, so let’s get at it now. Any version of SAS that can run ODS and the tagset ExcelXP will do for the SAS platform (V8 and above). Let’s look at the SAS Code in this section. This code creates a simple SAS File with one variable, one observation, and one value, the word “Hello”. This project is to get Excel to add the word “World” to the phrase, in a place known to Excel and visible to us. The rest of the code, opens the ODS Tagset as an output destination, and creates a *.xml file using the PROC PRINT procedure. Finally the “X” command is used to run Microsoft Excel and provide the new *.xml file as input to the Excel system. The ODS tagset generates an excel “sheet” name of “Greetings”, a background view in the Excel file with minimal formatting, and the font size of the headers will be a little larger than the rest of the information, and PROC PRINT will split the header “Greetings World” above the value “Hello” onto two lines.

CREATE A SIMPLE SAS FILE (THE HELLO PART OF THE PROJECT)

A Note about “Cut-and-Paste” – This code (or any SAS code) may not transfer from one format (i.e. *.PDF or *.DOC) to the same text characters that SAS uses. Special characters or Quotes may not convert correctly. The code may need to be entered or adjusted manually.

```

* Create a simple SAS File;
Data Hello_to_the_world;
  Greeting = 'Hello';
Run;

* Start ODS and name the output xml file;
%let hello_xml = c:\temp\Hello_world_test.xml;
Ods tagsets.excelxp file="&hello_xml";

* Setup ods options to call the proc print procedure;
Ods tagsets.excelxp options(Sheet_Name='Greeting') Style=minimal;

* Create an xml file using ods output and proc print;
Proc Print data = Hello_to_the_world label split = '*' noobs;
Label greeting = 'Greetings*World';
Var greeting / style(head) = {Font_Size = 2}; * increase row 1 font size;
Run;

Ods tagsets.excelxp close;

Options noxwait noxsync;
* run Excel 2003 - default path - pick one;
X "'C:\Program Files\Microsoft Office\OFFICE11\EXCEL.EXE' &hello_xml";

* run Excel 2007 - default path - pick one;
*X "'C:\Program Files(x86)\Microsoft Office\Office12\EXCEL.EXE' &hello_xml";

Options xwait xsync;
Run;

```

SETUP THE EXCEL ENVIRONMENT

That was the easy part, now on to the Excel setup. Most Excel users know that Excel will record a macro, and the macro can be associated with a key-stroke command that will run the macro. These macros are usually stored within the current workbook, or deleted before exiting Excel. What most users do not know is that Excel has a way to store macros so they are available to be used on all workbooks on a given computer, and not stored with the workbook. Furthermore, there is a way to give those macros control of Excel BEFORE the spreadsheets are visible to the user. The macros are stored in a “Start-Up” location that Excel loads into the Visual Basic Workbench before accessing the spreadsheets. These commands (Macros) are available when the workbooks start loading into Excel. The Windows XP default location for storing the system macros is the user accessible directory named **“C:\Documents and Settings\[your-user-id]\Application Data\Microsoft\Excel\XLSTART”**. And the Excel 97-2003 default file name is PERSONAL.XLS.

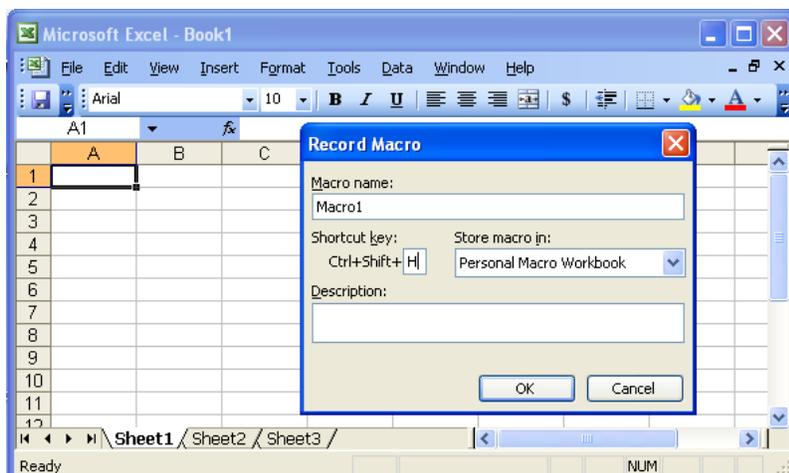


Figure 1. Selecting the Personal Macro Workbook to store an Excel macro.

When you create or update this Macro Workbook (it can hold data too) you will be prompted in the following way if the Personal Workbook has changes that need to be saved.

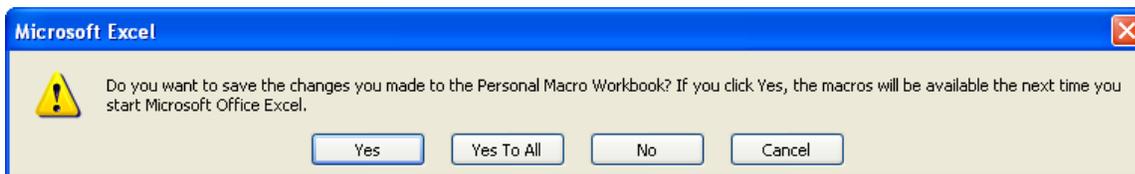
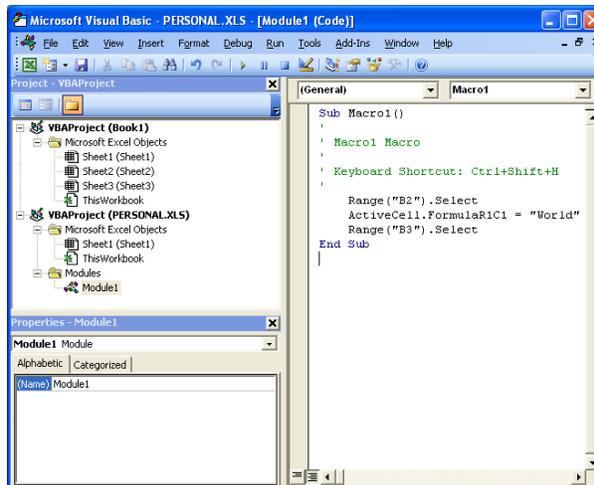
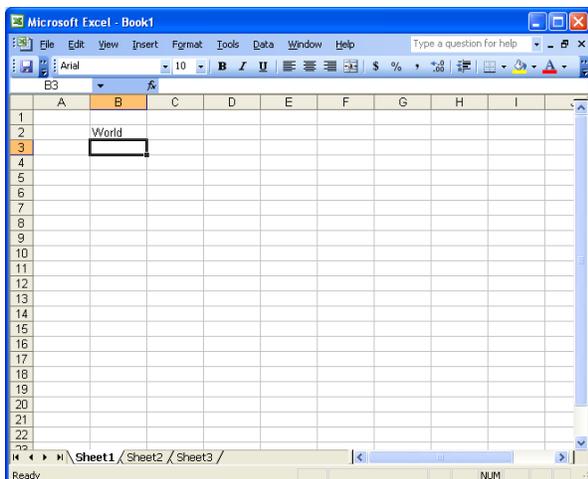


Figure 2. Prompt to save the PERSONAL.XLS (or PERSONAL.XLSB) macro workbook.

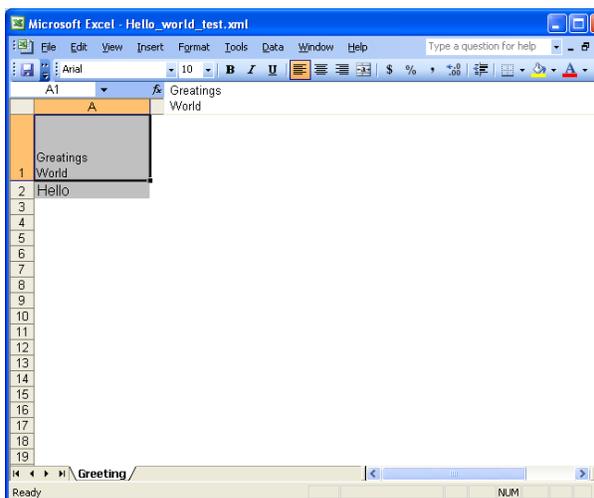
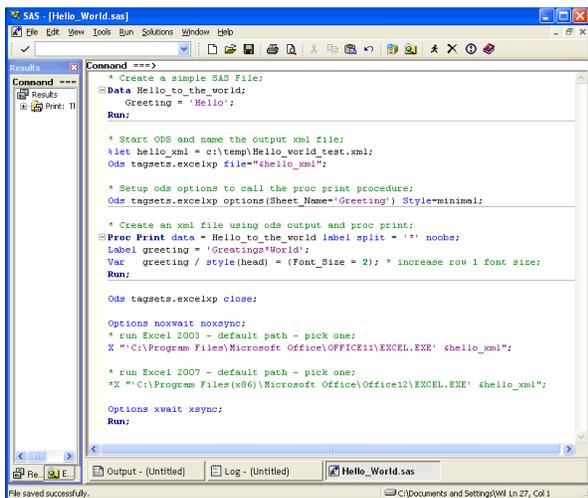
CREATE A SIMPLE EXCELFILE AND MACRO (THE WORLD PART OF THE PROJECT)

The next step is to record a macro, in this case we will continue with recording “Macro1” with the CNTL/SHIFT/H hot key assignment shown above. If we do this in a new workbook called “Book1” all that needs to be done is type “World” in cell “B2”. Then stop recording. The results look like the following two screen shoots.



Figures 3 and 4. Sample workbook and PERSONAL.XLS macro to write “World” in cell “B2”.

Now we have the beginning and the end of the project. So, running the SAS programs that will also run Excel and produce the following output.



Figures 5 and 6. SAS Code and the Excel system way that Excel looks when the *.xml file opens (Note that the heading “Greetings World” is on two lines in the Excel output file.

But to get the word “World” into cell “B2” the hot key CNTL/SHIFT/H still needs to be typed by the user.

THE FIX – ANOTHER EXCEL MACRO

This Excel macro is the key to the process. It resides in a special place. **The special place is**

“The First Place Given Control to the Programmer, by Excel”

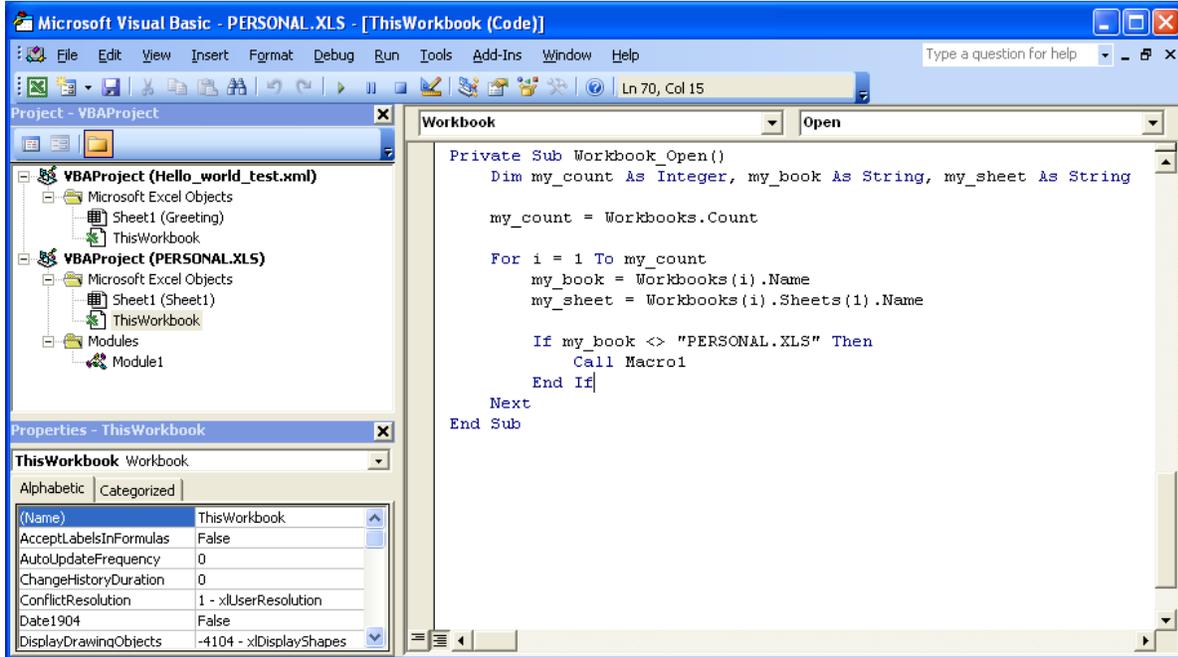


Figure 7. Workbook_Open Excel user written macro

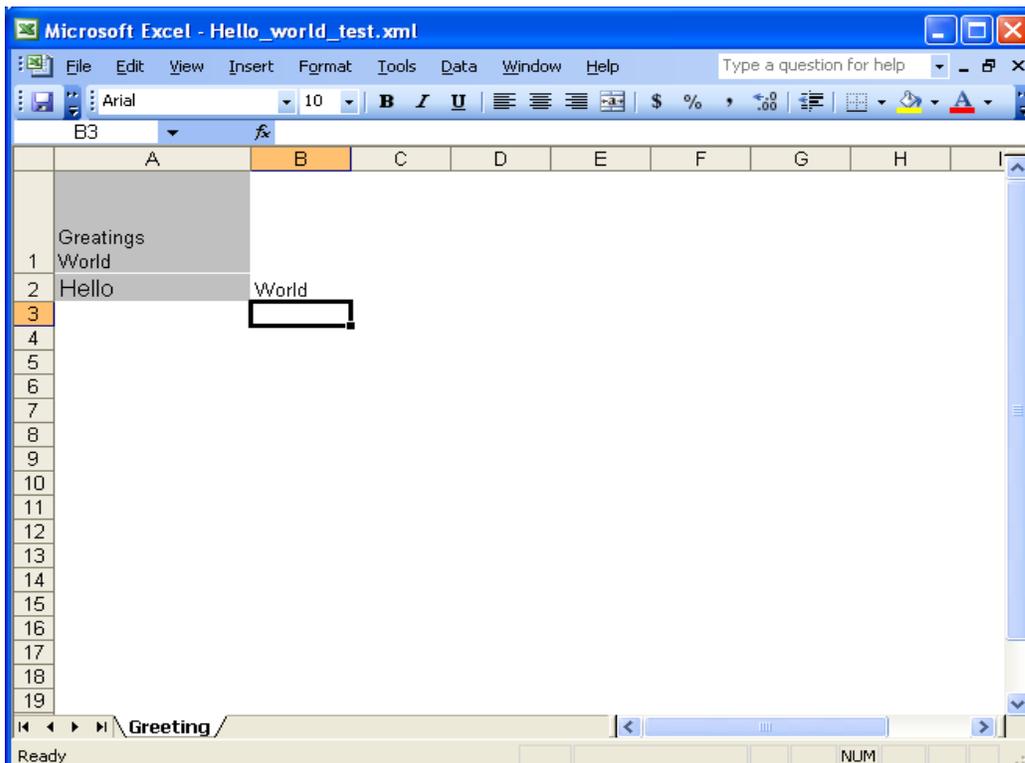


Figure 8. Result when SAS ran and opened Excel with the Workbook_Open Excel user written macro installed.

A NOTE ABOUT EXCEL

Like most other SAS Programmers that are also Excel users, this author has determined that Excel likes to dance to its own tune. Which means as soon as someone tells you what Excel will do in a particular situation; it will seem to do something else. This is the case with getting the macros listed here to execute correctly (meaning how you want them to execute, not how Excel thinks they should run). To that end, it should be pointed out that if you double click on an Excel file to open it, Excel may do its processing slightly differently than if the SAS system executes an “X” command to run Excel. Yes, that does not sound right, but try it for yourself. A clever little trick is to place a macro in the “**WORKBOOK_OPEN**” Excel macro with a message box to display information about what is going on at that instant. This will allow you to check for yourself. Remember our last macro in “**WORKBOOK_OPEN**” (here it is with an added “MsgBox” command. This neat little debugging tool can be useful in figuring out what is going on with Excel at the start of the program (when you cannot start the debugger) to give you guidance about how to process the code. The color is to highlight the code, it will not appear in red type in the macro workbook.

```
Private Sub Workbook_open()  
    Dim my_count As Integer, my_book As String, my_sheet As String  
    My_count = Workbooks.Count  
  
    Msg = "sub=workbook_open - workbook count= " + Str(my_count) 'Define message  
    Style = vbOKOnly 'Define buttons  
    Title = "Workbook name = " + ThisWorkbook.Name 'Define title  
    Response = MsgBox(Msg, Style, Title) 'wait for user  
  
    For I = 1 To my_count  
        my_book = Workbooks(i).Name  
        my_sheet = Workbooks(i).Sheets(1).Name  
  
        Msg = "sub=workbook count loop - workbook counter= " + Str(i)  
        Title = "Workbook name = " + my_book  
        Response = MsgBox(Msg, Style, Title)  
  
        if my_book <> "PERSONAL.XLS" Then  
            Call Macro1  
        End If  
    Next  
End Sub
```

AN EXPLANATION OF THE “WORKBOOK_OPEN” USER WRITTEN EXCEL MACRO

It may seem a little more complex than needed, it is a little (but the Message Boxes can go away). If the macro had been a simple call to “**Macro1**” you might think it would have worked, but remember that now your computer has a new “**Hidden Workbook**” that Excel will always (yes always) open. Look at the Excel VBA desktop above, on the left side it shows TWO workbooks open, “**Hello_World_Test.xml**” and “**PERSONAL.XLS**”. One thing that has been relatively consistent is that when SAS uses the “X” command both workbooks will be open (“**Hello_World_Test.xml**” and “**PERSONAL.XLS**”) when the “**WORKBOOK_OPEN**” macro runs. This will allow the “**Macro1**” code to execute. However, a double click on an Excel file icon, will usually only have one workbook open, then the Excel macro “**WORKBOOK_OPEN**” runs (that workbook us the “**PERSONAL.XLS**” macro).So we will assume that for the purposes of this paper all calls to Excel come from SAS and the “X” command.

A STRONGER “WORKBOOK_OPEN” EXCEL MACRO

Since we do not want every Excel workbook to have “World” in cell “B2” the next step is to enhance the Excel macro “**WORKBOOK_OPEN**” so that it is more selective when it runs. So, we will take out the message boxes and add in control language to interpret information sent from SAS. By sending commands to Excel from SAS we can control what actions our Excel macro takes when it gets control of the system.

Let us upgrade our SAS code first.

```
* Total sales by region, and put it into a file;  
Proc summary data= SASHELP.Shoes nway;  
weight sales;  
var sales;  
output out=sales(where=( _stat_ = "SUMWGT" ));  
by region;
```

```

run;

*****;
** start ODS and name the output xml file          **;
*****;
%let xml_file = C:\Excel_book\Excel_Data\Pie_Chart_test.xml;

ods tagsets.excelxp file="&xml_file." ;

*****;
* Set up the ExcelXP options then run the Proc Print routine ;
*****;
ods tagsets.excelxp options(Sheet_Name='Sales') Style=Minimal ;

*****;
** Run Proc print using labels and noobs.          **;
*****;
proc print data = sales noobs label;
label region = "?graph_1?Region"; * add control value for Excel to use here;
var  region sales ;
run;

*****;
** Clean up and Call Excel to open the new *.XML file*;
*****;
ods tagsets.excelxp close;
options noxwait noxsync;
* run Excel 2003;
x "C:\Program Files\Microsoft Office\Office11\excel.exe' &xml_file";
run;

```

When we run this SAS code and send the data to Excel with our new “**WORKBOOK_OPEN**” it looks something like this:

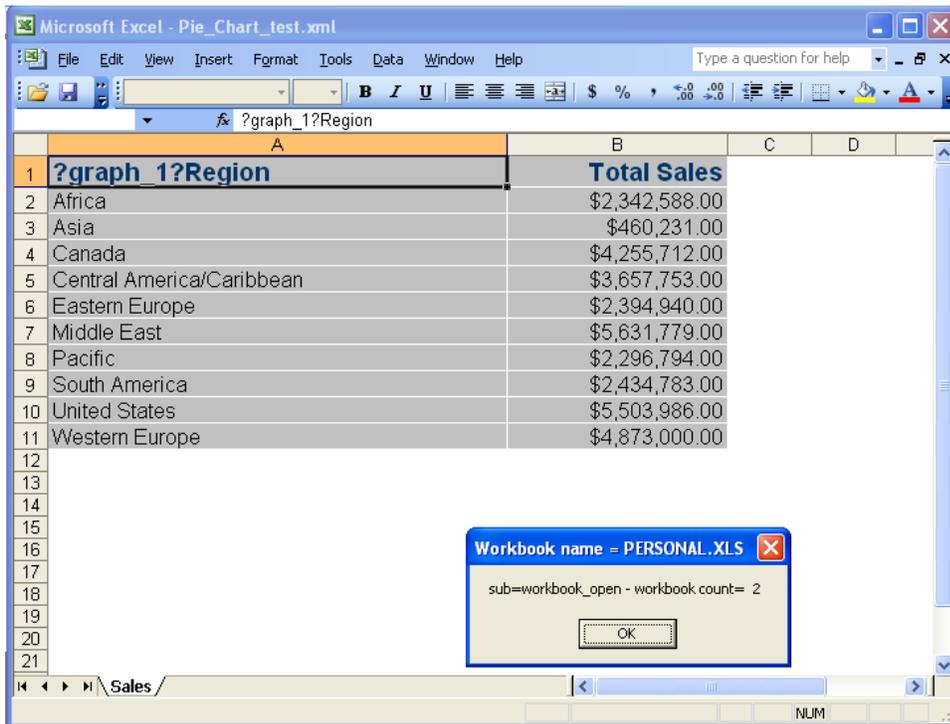
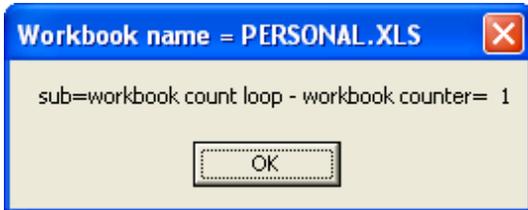


Figure 9. SAS output displayed in Excel, and first message box from Excel.

Now the new "WORKBOOK_OPEN" macro

Our little message box is doing its job, as you can see it has told us that the currently open workbook is the "PERSONAL.XLS" workbook. The first and second clicks on "OK" gives the following message boxes. (cutting and pasting the code may not convert the quote marks correctly, so just retype them).



Figures 10 and 11. Testing the "PERSONAL.XLS" workbook macro, after clicking "OK" to continue.

Also notice that cell "B2" now contains the word "World" which appeared after our macro1 code ran.

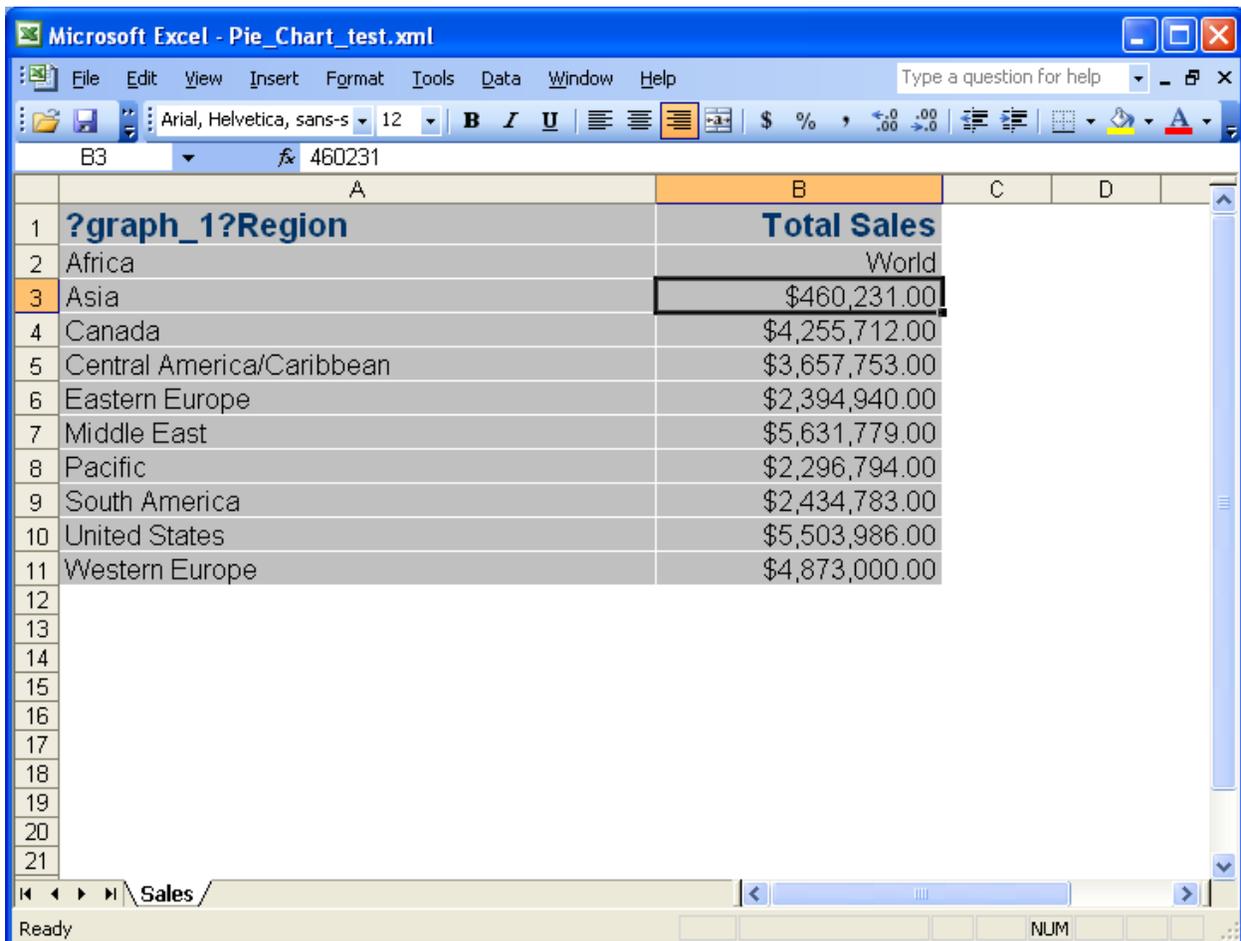


Figure 12. Excel workbook after running SAS to open the Workbook and run the Macro1 code. Note the word "World" in cell "B2".

Since we do not want to change cell “B2” to the word “World” for every workbook we create, lets change the macro code for the “WORKBOOK_OPEN” macro. Remember the control information we inserted into the label of the “Region” variable? Well look at cell “A1”. The code in “red” below is going to now use that information to “Command” Excel to do what we want it to do now. (note the comments below might not “Cut-n-Paste” well).

```

Private Sub Workbook_Open()
    Dim my_count As Integer, my_book As String, my_sheet As String
    my_count = Workbooks.Count

    For i = 1 To my_count
        my_book = Workbooks(i).Name
        my_sheet = Workbooks(i).Sheets(1).Name
        If my_book <> "PERSONAL>XLS" Then

            my_cell_a1 = Workbooks(i).Sheets(1).Range("A1").Value
            my_cell_size = Len(my_cell_a1)

            ' look for a second ? from right side of value
            my_flag_size = InStrRev(my_cell_a1, "?") - 1

            ' if my_flag_size is greater than 1 then two ? were found
            ' this is a special processing workbook
            ' so get the info and process the workbook
            ' else ignore and return control to Excel and the user
            If my_flag_size > 1 Then

                ' get left ? character and data between the two ?'s
                my_report = Left(my_cell_a1, my_flag_size)

                ' remove the left ?
                my_data_size = Len(my_report) - 1
                my_report = Right(my_report, my_data_size)

                ' remove control characters from Cell A1 value
                ' and store in cell A1
                Workbooks(i).Sheets(1).Range("A1").Value = _
                    Right(my_cell_a1, (my_cell_size - (my_flag_size + 1)))

                ' add new cases to process new reports
                Select Case my_report

                    Case "graph_1"
                        Call Module1.my_graph_1(my_sheet)
                    Case Else

                End Select

            End If
        End If
    Next
End Sub

```

Now the new "MODULE1.MY_GRAPH_1" macro

```
Sub my_graph_1(my_sheet As String)
'
' my_graph_1 Macro
'
' This macro generates a pie chart with minimal number of extras
' The code is reproduced here exactly as it was produced by the
' Excel Macro Record feature -
' except the name values "my_sheet" and "Chart 1" were adjusted
'

Range("A1:B11").Select
Charts.Add
ActiveChart.ChartType = xl3DPie
ActiveChart.SetSourceData Source:=Sheets(my_sheet).Range("A1:B11"), PlotBy _
    :=xlColumns
ActiveChart.Location Where:=xlLocationAsObject, Name:=my_sheet
With ActiveChart
    .HasTitle = True
    .ChartTitle.Characters.Text = "My_new_Pie_chart"
End With
ActiveChart.HasLegend = True
ActiveChart.Legend.Select
Selection.Position = xlBottom
ActiveChart.Legend.Select
Selection.AutoScaleFont = True
With Selection.Font
    .Name = "Times New Roman"
    .FontStyle = "Regular"
    .Size = 11
    .Strikethrough = False
    .Superscript = False
    .Subscript = False
    .OutlineFont = False
    .Shadow = False
    .Underline = xlUnderlineStyleNone
    .ColorIndex = xlAutomatic
    .Background = xlAutomatic
End With
Selection.Position = xlLeft
ActiveChart.ChartArea.Select
With ActiveChart
    .Elevation = 35
    .Perspective = 30
    .Rotation = 0
    .RightAngleAxes = False
    .HeightPercent = 100
End With

' changed "Chart 1" to the numeric location of the chart - not the chart name
ActiveSheet.Shapes(1).ScaleHeight 1.25, msoFalse, msoScaleFromTopLeft
ActiveSheet.Shapes(1).ScaleHeight 1.5, msoFalse, msoScaleFromBottomRight

'ActiveSheet.Shapes("Chart 1").ScaleHeight 1.25, msoFalse, msoScaleFromTopLeft
'ActiveSheet.Shapes("Chart 1").ScaleHeight 1.5, msoFalse, msoScaleFromBottomRight

Application.CommandBars("Task Pane").Visible = False
End Sub
```

Below is a simple graph generated by the macro `"my_graph_1"` when SAS issued a "X" command to start Excel. The SAS code on the next line selected the routine to execute in Excel.

label region = "?graph_1?Region"; * add control value for Excel to use here;

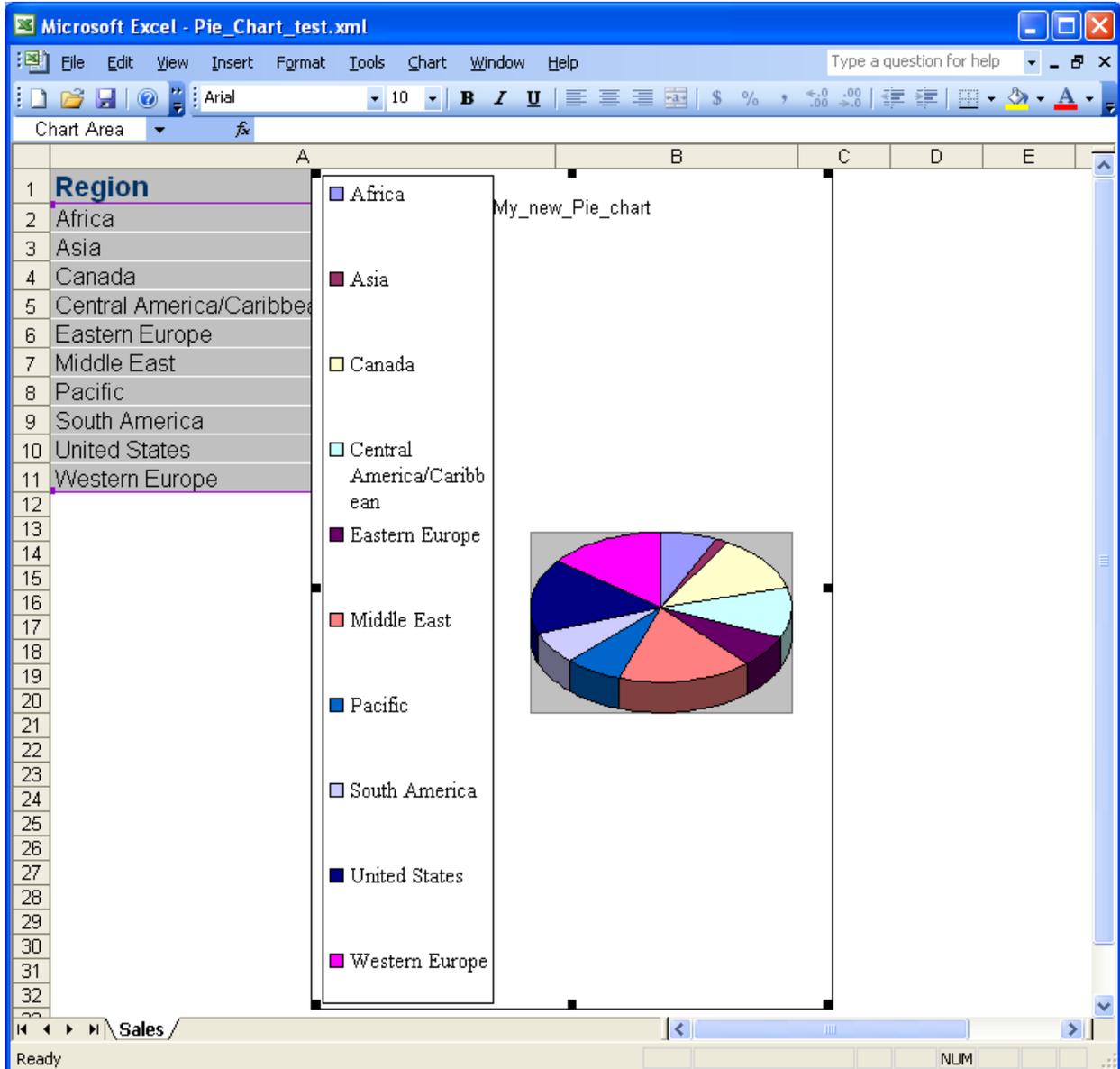


Figure 13. Simple graph generated by Excel when SAS issued the "X" command. The source data for the graph is hidden behind the graph in Column "B".

let's see what will happen if we change the SAS label statement to something different ... Like the following:

label region = "?graph_2?Region"; * add control value for Excel to use here;

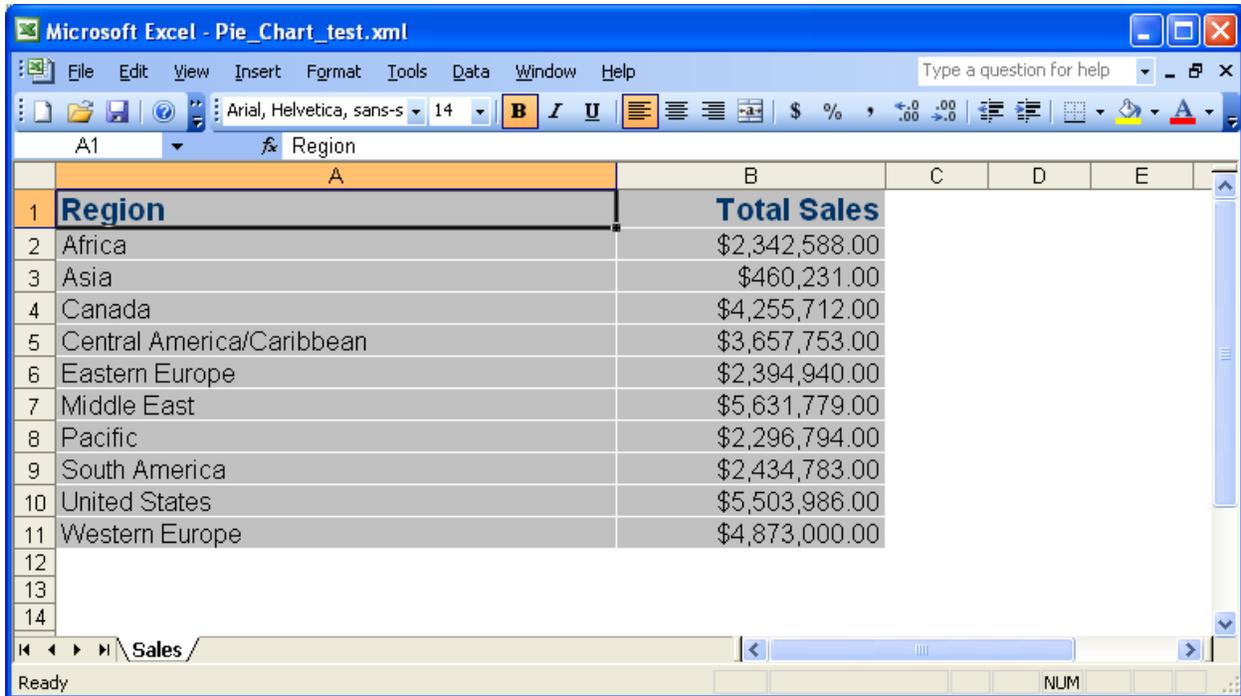


Figure 14. Results when LABEL REGION = "?GRAPH_2?REGION". (Note the control information was removed)

Figure 14 looks just like a regular Excel Spreadsheet, no graph, no funny characters in cell "A1" and in fact if we had changed the label to REGION = "?GRAPH_1 - REGION" we could have kept everything.

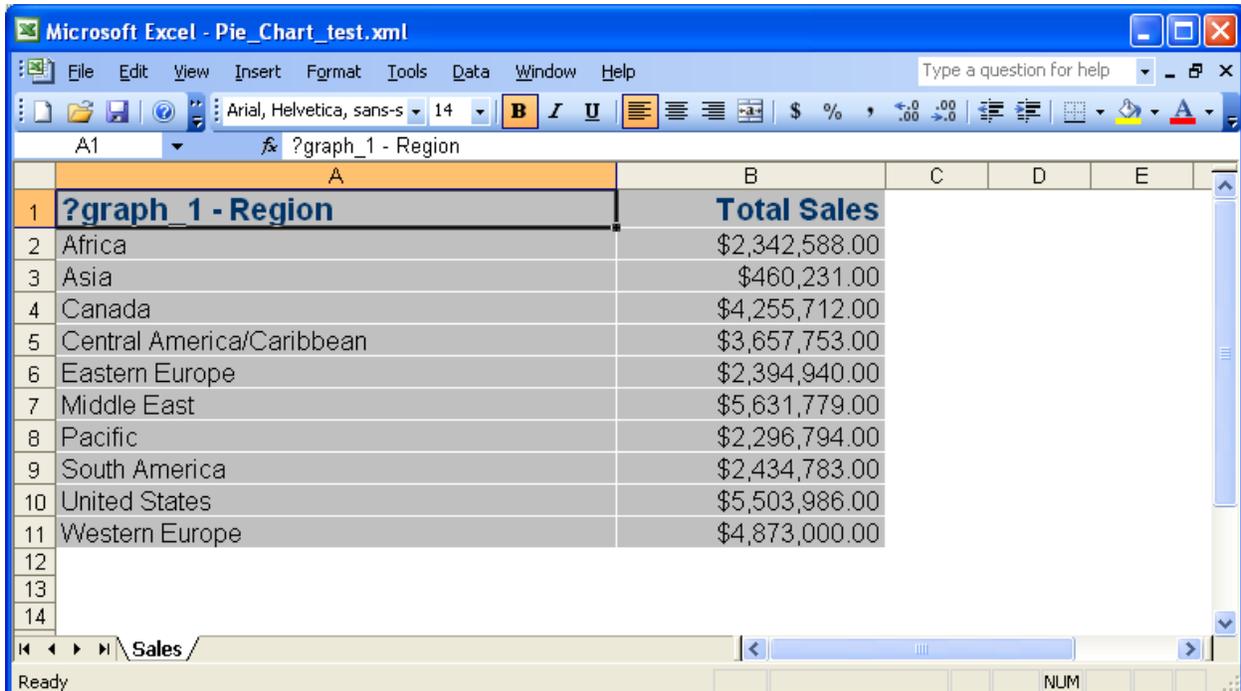


Figure 15. Sample with no control string in cell "A1". **NOTE** – Cell "A1" did not have two question marks in the cell and therefore our "Workbook_Open" macro ignored the contents and did not modify the cell.

CONCLUSION

This paper, as a proof of concept paper, has shown that it is possible with relatively simple tools to extend the reach to the programmers ability to automate output. It is this author's hope that this process can be an insight into the possibilities that are available. Of course any Excel code can be accessed using this method, and this truly means any code. Excel uses Visual Basic for Applications code and has access to routines that can output any file format that Excel can write. The user coded macros can also include code to read files, run "Microsoft Word" or "Microsoft Access" applications. With proper control structures the formatting of multiple page workbooks can be automated, Page titles can be added, column formatting, highlighting can be done. And of course anyone can make a better graph.

In the body of the paper a comment was made about the fact that Excel was "Always" going to open the "**PERSONAL.XLS / .XLSB**" workbook. Well that is only true if the workbooks are left in the "**XLSTART**" directory. To be able to say "**Hello World**" is a very powerful thing, because now the door is open to anything that BOTH SAS and EXCEL can do together.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name	William E Benjamin Jr
Enterprise	Owl Computer Consultancy, LLC
Work Phone:	602-942-0370
Fax:	602-942-3204
E-mail:	William@OwlComputerConsultancy.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.