

Paper 013-2012

Yes! SAS® ExcelXP WILL NOT Create a Microsoft Excel Graph, but SAS Users Can Command Microsoft Excel to Automatically Create Graphs from SAS ExcelXP

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

ABSTRACT

The SAS ODS Tagset ExcelXP creates *.xml output, and *.xml output cannot contain graphs. So how can SAS programmers get graphs into your Excel workbooks? One way is to build them in Excel yourself. This paper shows you how to create data using SAS, and then command Microsoft Excel to read the data, create a graph or fully reformat a worksheet, without putting an Excel macro into the output Excel Workbook. And the program will do it all while you watch, including for multiple sheets in a workbook. The SAS code, and Excel code, shown is a fully integrated system to create and format macro-free Excel workbooks, using SAS® 9 (SAS® 8 if Internet downloads are available) and Excel 97 and above.

KEYWORDS

Base SAS, EXCEL, VBA Macro Code, Automated EXCEL Formatting, ODS, XML files

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 or location 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 listed below on the next page. This code also appears in Figure 5. This code does the following:

- 1) Create 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.
- 2) Open an ODS Tagset as an output destination, and create a *.xml file using the PROC PRINT procedure.
- 3) PROC PRINT will split the header “Greetings World” above the value “Hello” onto two lines.
- 4) Run Microsoft Excel and provide the new *.xml file as input to the Excel system.
- 5) The ODS tagset generates an excel “sheet” with the name “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.

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 or quoted information may need to be entered 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. If the directory does not exist in this location on your system ask your system administrator for the exact location of the Excel directory.

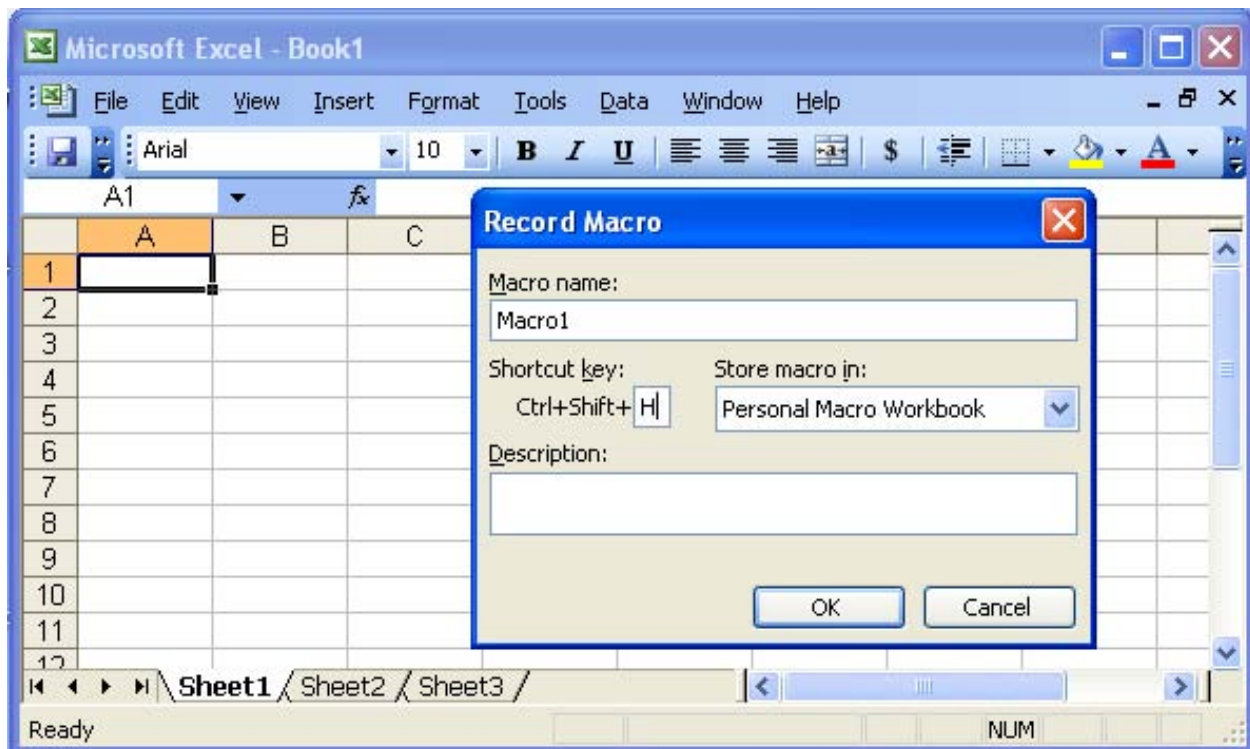


Figure 1. Selecting the Personal Macro Workbook to store an workbook with personal Excel macros.

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. As mentioned above this workbook can store many macros and make them available every time you open Excel. When you no longer need to access these macros locate and delete the PERSONAL.XLS (or PERSONAL.XLSB) file in your XLSTART directory mentioned above.

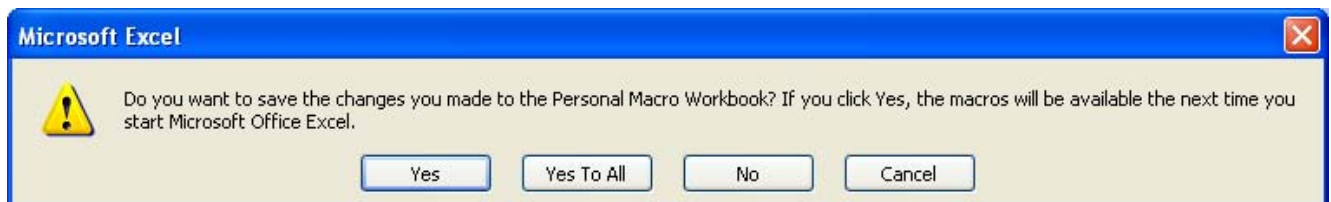
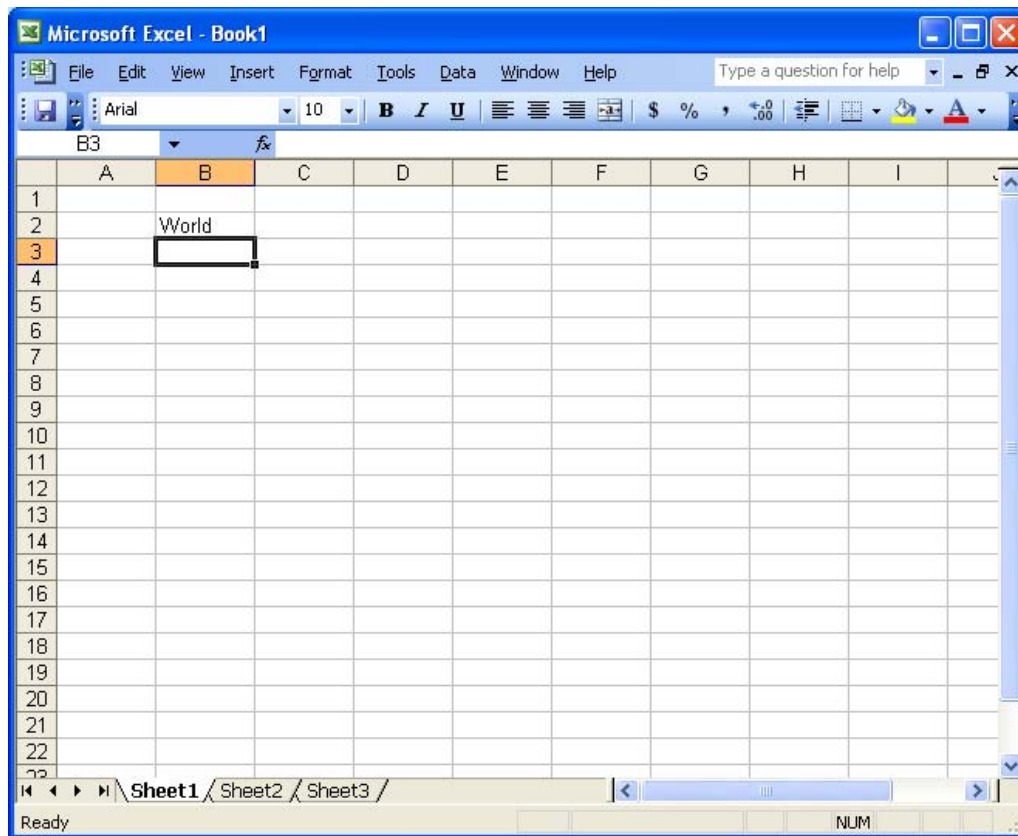


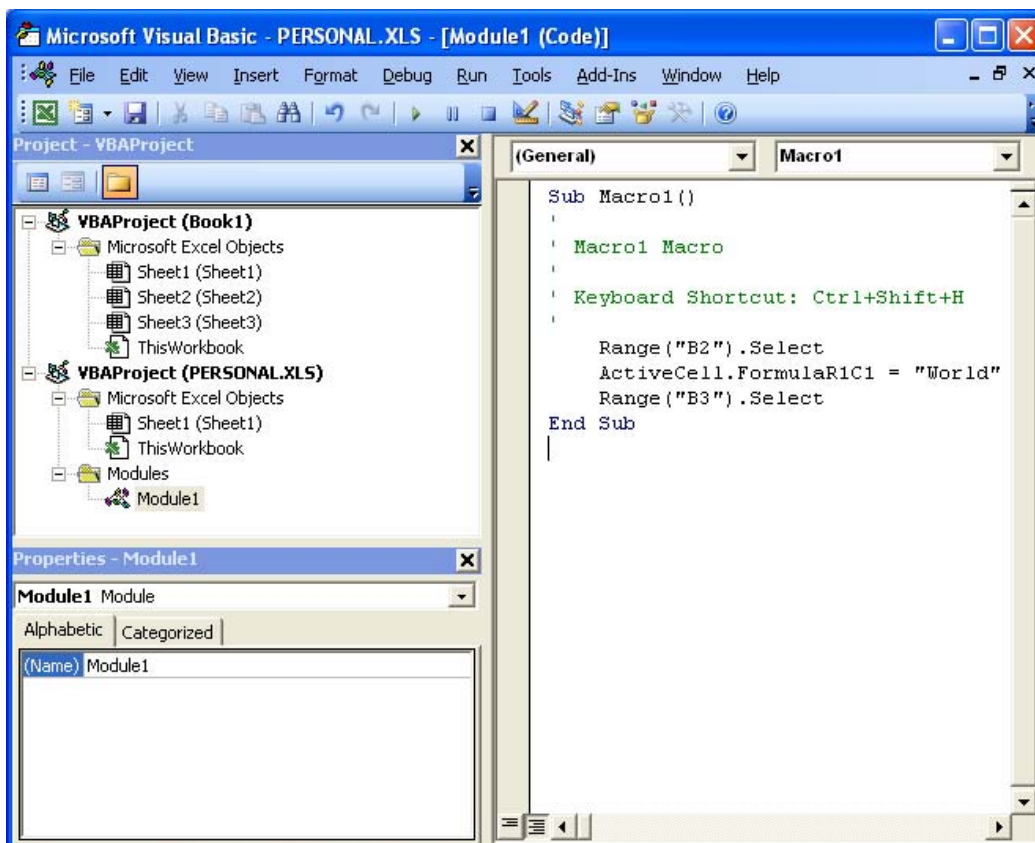
Figure 2. Prompt to save the PERSONAL.XLS (or PERSONAL.XLSB) macro workbook.

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 CTRL/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. When we choose the "Personal Macro Workbook" as shown above the output macro is stored as a macro in a module, in this case called "Macro1" in "Module1". This macro will be stored in a hidden workbook called PERSONAL.XLS (or PERSONAL.XLSB) in your XLSTART directory.

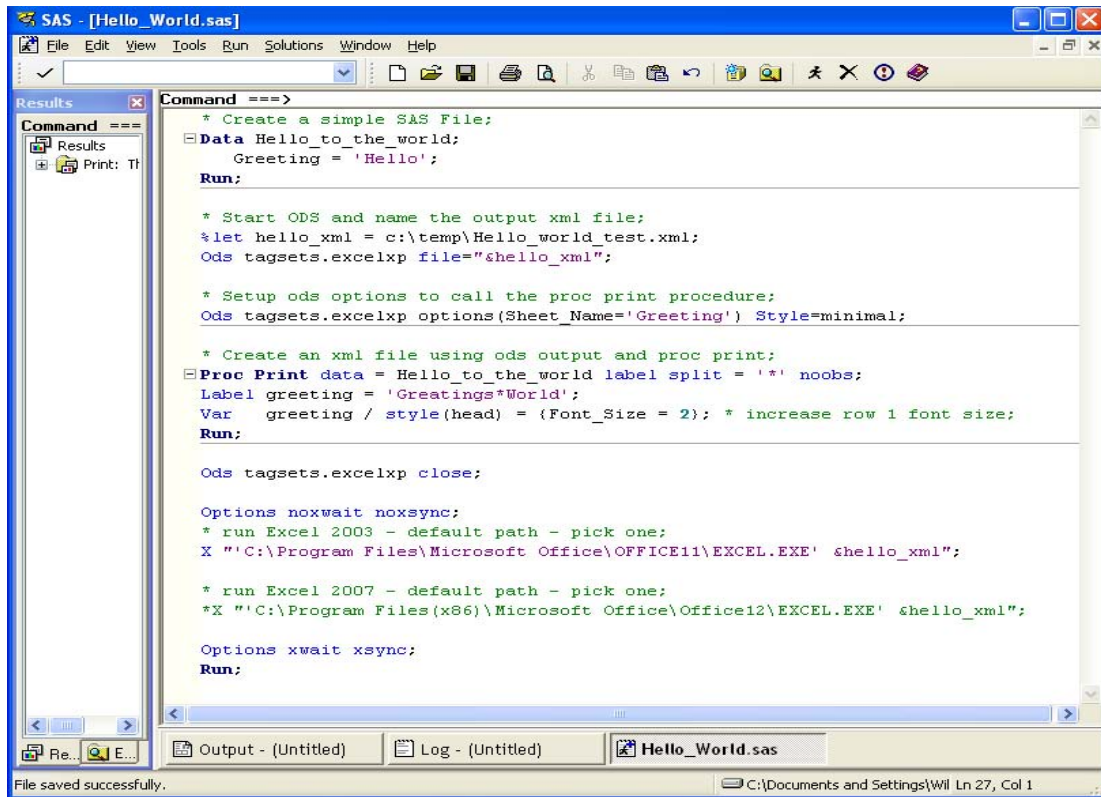


Figures 3. This figure shows the result of executing the macro by typing CTRL/SHIFT/H.

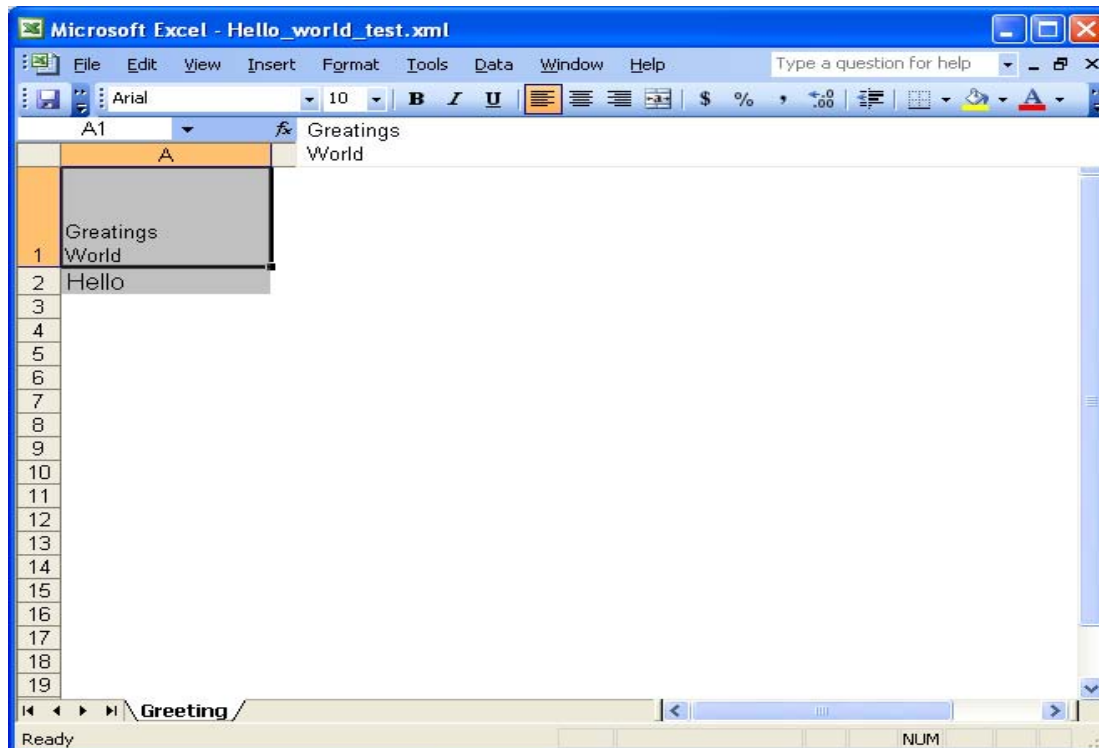


Figures 4. Sample workbook and PERSONAL.XLS macro to write "World" in cell "B2".

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



Figures 5. SAS Code to generate the "Hello_World_test.xml" file and execute Excel.



Figures 6. The way that Excel looks when the *.xml file opens (Note that the heading "Greetings World" is on two lines in the Excel output file.

We are almost there, but to get the word “World” into cell “B2” the hot key CTRL/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”

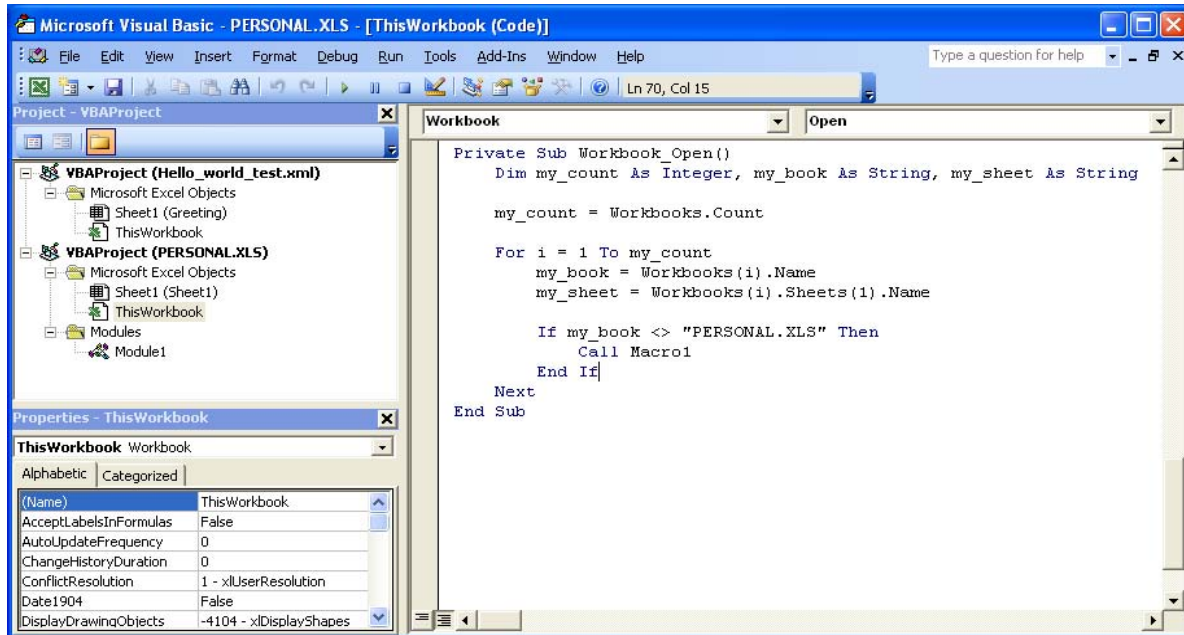


Figure 7. Workbook_Open Excel user written macro, stored in the “This Workbook” area of PERSONAL.XLS.

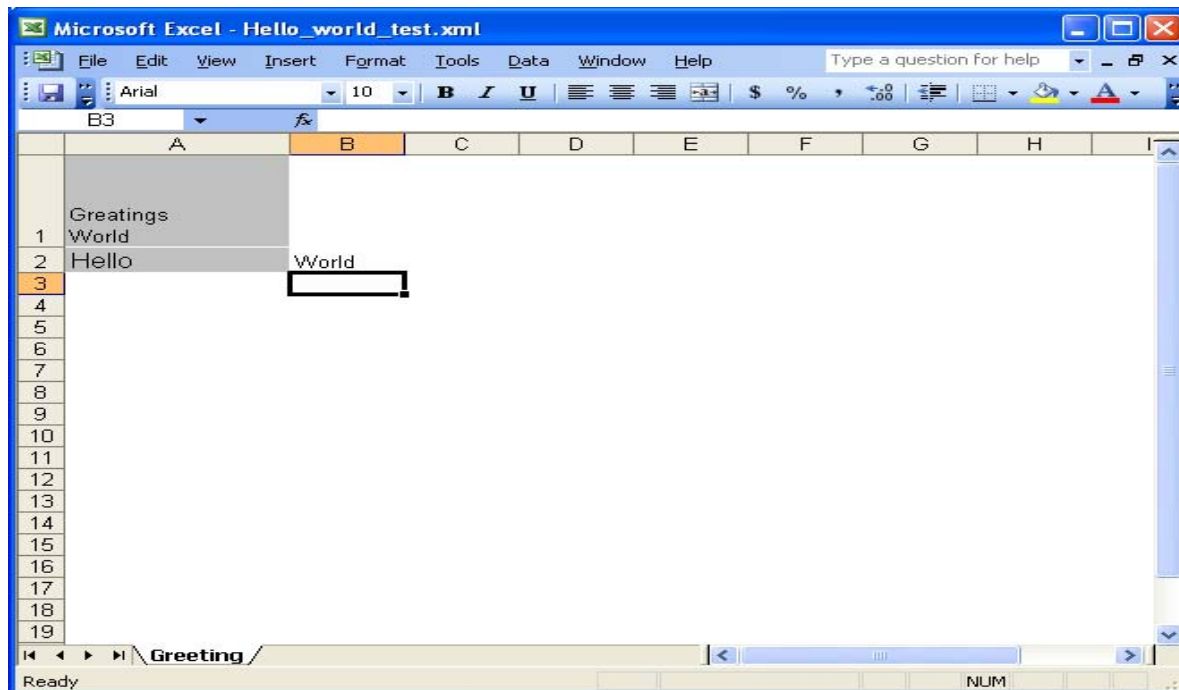


Figure 8. Result when SAS ran and opened Excel with the Workbook_Open Excel user written macro installed.

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 in Figure 7 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 is 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;

```

This SAS Code contains something we have not seen yet. Within the proc print “LABEL” command (label region = "?**graph_1?Region**";) there is embedded information we want to pass to Excel. The object is to send information to Excel to tell Excel what to do next. It is nice to have a constant location to keep it simple for the Students (KISS). This author always uses cell “A1” of the first sheet of the Excel workbook, and brackets the code with characters not normally found in titles or variable names. This can also be done by changing a variable name to `_MyCode_AndVariableName` for the first column of the first sheet. (the **BOLD** type is only to highlight the command and will not appear in the SAS code or Excel workbook)

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

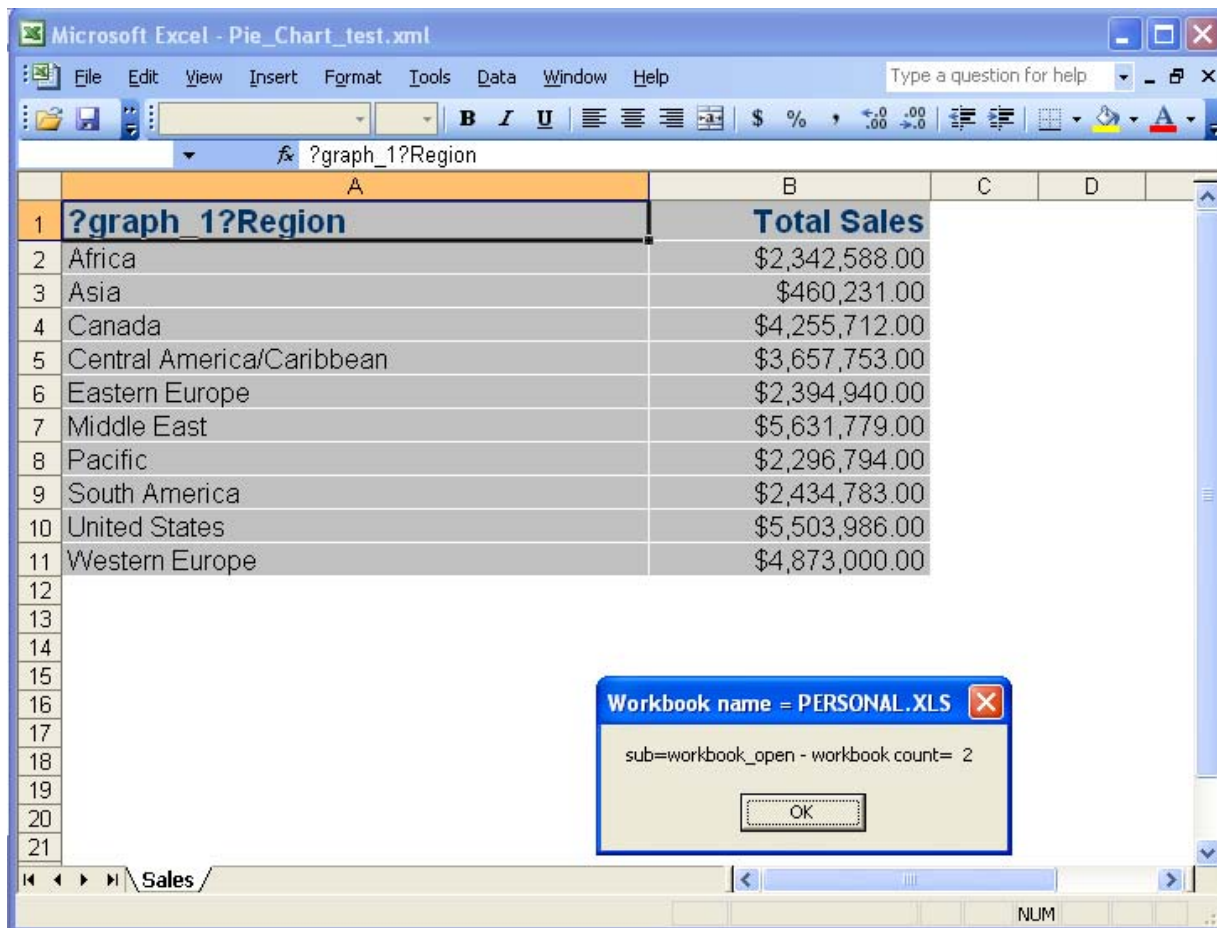


Figure 9. SAS output displayed in Excel.

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” give the following message boxes. (cutting and pasting the code may not convert the quote marks correctly, so just retype them).

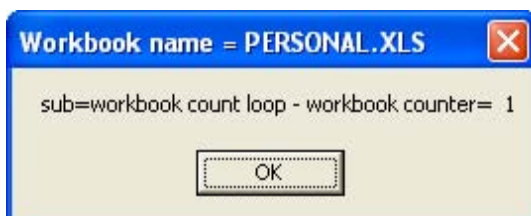


Figure 10. Testing the “PERSONAL.XLS” workbook macro.

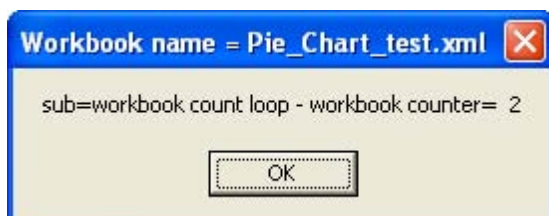
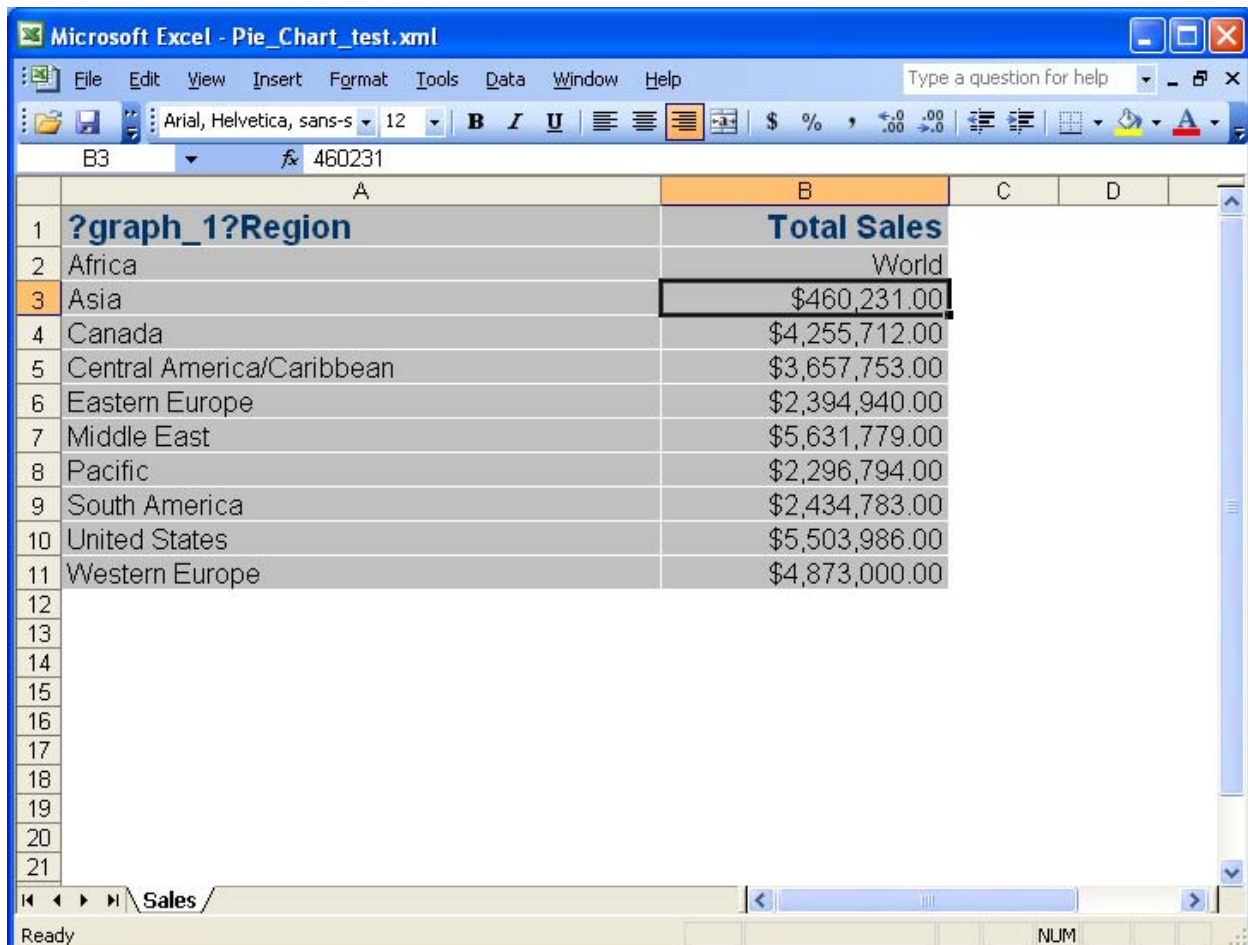


Figure 11. Testing the “PERSONAL.XLS” workbook macro.

Also notice that cell “B2” still contains the word “**World**” which appeared after out macro1 code ran.



The screenshot shows an Excel window titled "Microsoft Excel - Pie_Chart_test.xml". The active sheet is "Sales". The table has two columns: "Region" (A) and "Total Sales" (B). The data is as follows:

Region	Total Sales
Africa	World
Asia	\$460,231.00
Canada	\$4,255,712.00
Central America/Caribbean	\$3,657,753.00
Eastern Europe	\$2,394,940.00
Middle East	\$5,631,779.00
Pacific	\$2,296,794.00
South America	\$2,434,783.00
United States	\$5,503,986.00
Western Europe	\$4,873,000.00

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

Now we have something truly automated, even though we still have to press enter when the text boxes appear. We now have a procedure that starts by running SAS Code and ends by displaying an active Excel workbook that not only has our data, but has made a change to our data. But, since we do not want to change cell “B2” to the word “**World**” for every workbook we create, let’s 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. The source of this code is the original “`Workbook_Open()`” from Figure 7 above. Excel VBA Code uses the single quote “**'**” (shown here larger than normal) to identify comments, and the double quotes to enclose text. The code on the next page below locates paired question marks in cell “A1” of the first sheet of the workbook and places the text between the question marks into an EXCEL variable called “`my_report`” that is used to control selection of a macro to process. The Excel VBA “`Select Case`” code maps directly to the SAS “`Select`” statement and Excel is case sensitive.

```
Select Case my_report
Case "graph_1"
    Call Module1.my_graph_1(my_sheet)
Case Else
End Select
```

(note the comments below might not “Cut-n-Paste” well because of the quote characters)

```

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

See appendix A for a detailed explanation of the EXCEL VBA Code for Workbook_Open() and my_graph_1() .

```
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 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;

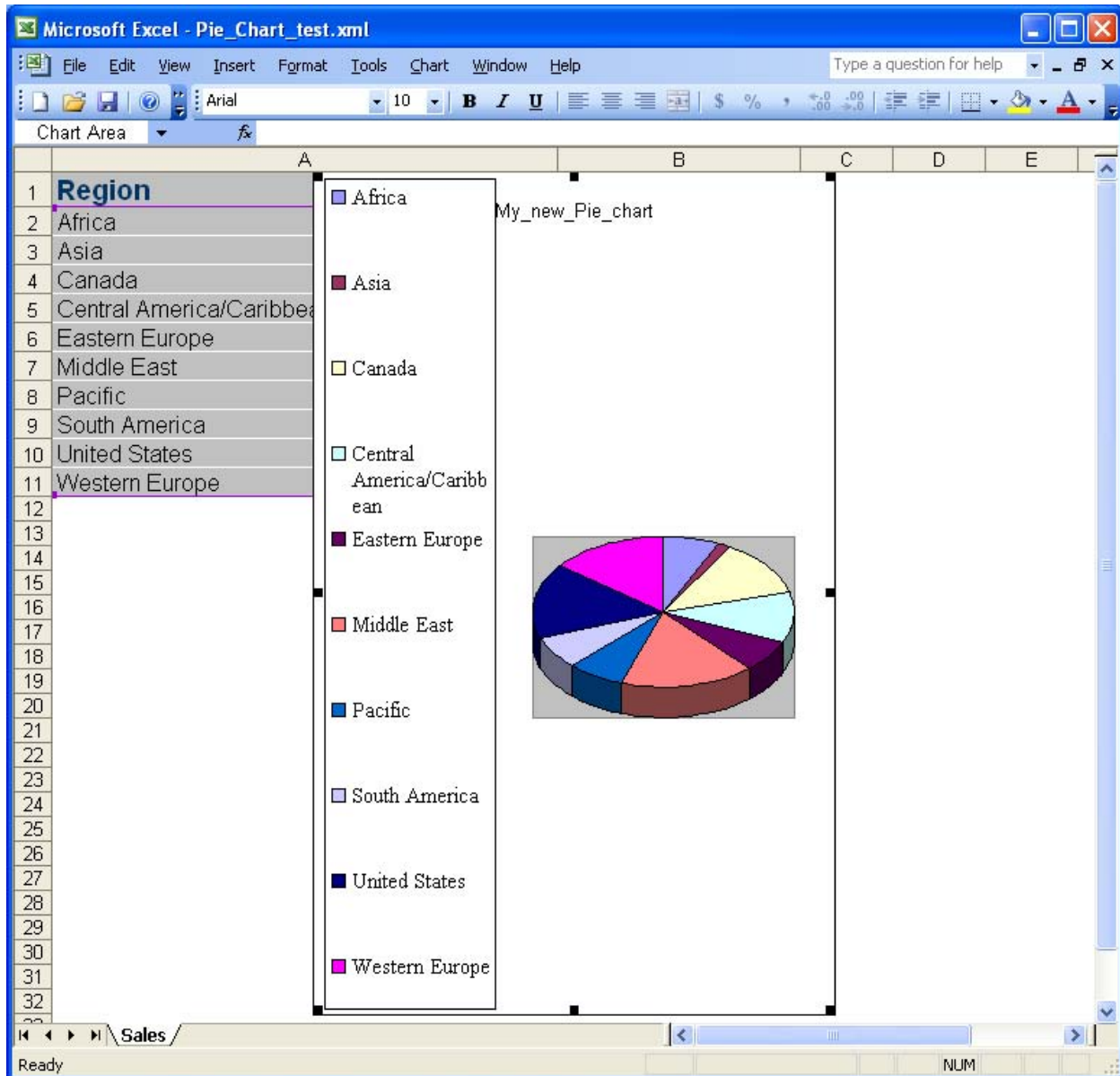


Figure 13. Simple graph generated by Excel when SAS issued the "X" command.

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;

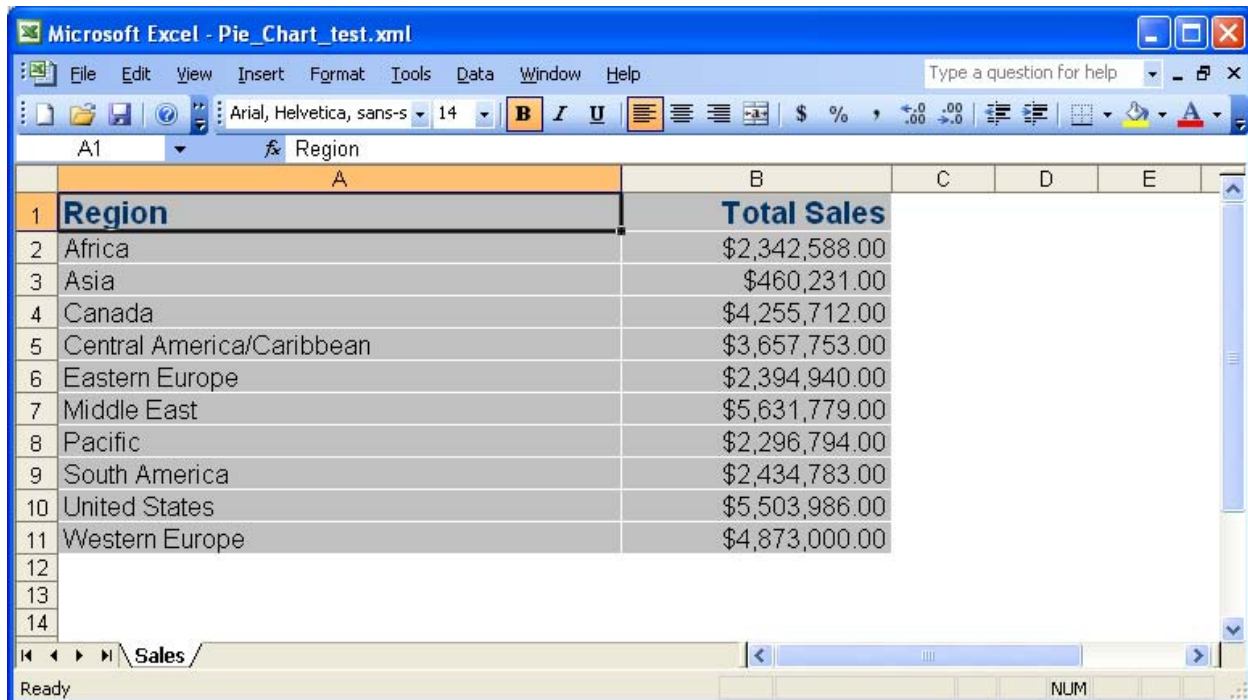


Figure 14. Results when LABEL REGION = "?GRAPH_2?REGION". (Note the control information was removed)

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.

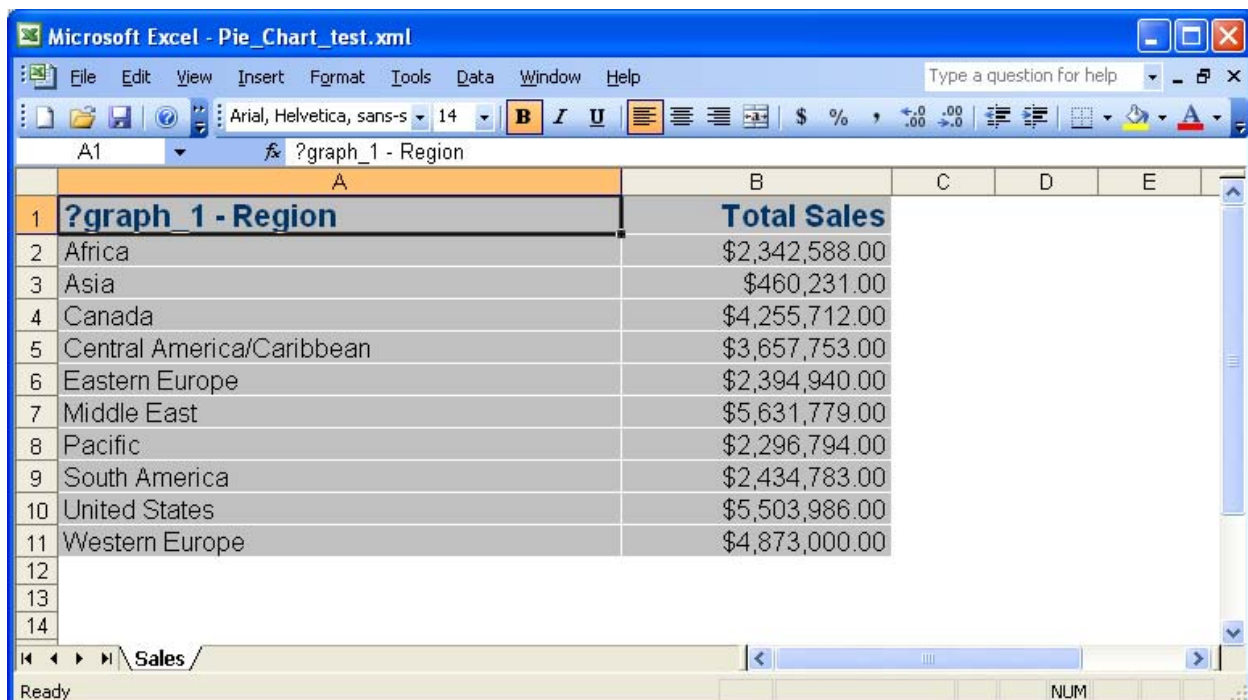


Figure 15. Sample with no control string in cell "A1".

CONCLUSION

This paper, as a proof of concept paper, has shown that it is possible with relatively simple tools to extend the reach of the programmer's 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.

REFERENCE MATERIAL INFORMATION

The only reference materials used to develop this paper were the SAS and Microsoft help files provided with their respective products.

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.

Appendix A – Two Excel VBA code Routines

'Editor Notes here are prefixed with '(A)', '(B)', '(C)', etc. and will appear as comments in the Excel Code Window:

'(A): The name of this macro is Workbook_Open. It is a Private subroutine stored in the VBAPROJECT area of the

'(A): workbook called "**PERSONAL.XLS**" (Excel 2003) or "**PERSONAL.XLSB**" Excel 2007/10. This

'(A): subroutine has no parameters.

```
Private Sub Workbook_Open()
```

'(B): This subroutine defines several local variables

'(B): my_count As an Integer variable

'(B): my_book As String variable

'(B): my_sheet As String variable

```
Dim my_count As Integer, my_book As String, my_sheet As String
```

'(C): The variable my_count is set to the number of workbooks that EXCEL currently has open.

```
my_count = Workbooks.Count
```

'(D): This VBA "For" loop (similar to the SAS "DO" loop) will execute once for each open workbook

```
For i = 1 To my_count
```

'(E): Information about workbooks and sheets are stored in arrays, here my_book is being set equal to the name

'(E): of the currently open workbook

```
my_book = Workbooks(i).Name
```

'(F): Information about workbooks and sheets are stored in arrays, here my_sheet is being set equal to the name

'(F): of the currently active worksheet

```
my_sheet = Workbooks(i).Sheets(1).Name
```

'(G): The name of the workbook is being tested to make sure it is not "**PERSONAL.XLS**", if it is no action is taken.

'(G): If the workbook name test is false then the rest of the if condition is executed (like in SAS)

```
If my_book <> "PERSONAL.XLS" Then
```

'(H): The contents of Cell "A1" of sheet one of the current user workbook is placed into the EXCEL variable

'(H): my_cell_a1

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

'(I): The number of characters that are in EXCEL variable my_cell_a1 are counted and placed into my_cell_size

```
my_cell_size = Len(my_cell_a1)
```

'(J): The EXCEL variable my_cell_a1 is tested for a "?" by looking at all of the characters in the variable except the

'(J): first character and loading the location of the "?" into the variable my_flag_size.

```

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

```

- '(K): if the length of my_flag_size is greater than 1 then there is more than one character between the two
 '(K): question marks on the left side of cell A1 of the worksheet. If there is no question mark on the left side
 '(K): of the cell A1 value then the value of variable my_report generated later will not match a "CASE" test value

```

.....
' 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

```

- '(L): Start building the variable my_report by getting the left part of the value without the question mark

```

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

```

- '(M): Calculate the size of the temporary value of my_report

```

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

```

- '(N): get the final contents of the variable my_report with no question marks – this removed the left question mark

```

my_report = Right(my_report, my_data_size)

```

- '(O): return the original value of cell A1 to the workbook cell A1 without the left most control characters. NOTE the
 '(O): underscore "_" on the line that begins with Workbooks(i)... is a continuation character so that command
 '(O): spreads over two lines.

```

' 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)))

```

- '(P): This "Select Case" command works much like a SAS "SELECT" statement, "Select Case my_report" is
 '(P): the same as "Select When(my_sas_variable_name)"

```

' add new cases to process new reports
Select Case my_report

```

- '(Q): The 'Case "graph_1"' value and the next line are the same as the 'When("value") SAS_command...;' code.

```

Case "graph_1"
Call Module1.my_graph_1(my_sheet)

```

- '(R): The "Case Else" command is the same as the SAS "Otherwise" command.

```

Case Else

```

- '(P): The end of the Select Case code

```

End Select

```

- '(K): The end of the if statement

```

End If

```

- '(G): The end of the if statement

```

End If

```

- '(D): The end of the for statement

```

Next

```

- '(A): the end of the subroutine

```

End Sub

```

'Editor Notes here are prefixed with ' [A], [B], [C], etc. and will appear as comments in the Excel Code Window:

[A]: The name of this macro is my_graph_1. It is a public subroutine stored in the MODULES area of the
 [A]: workbook called "**PERSONAL.XLS**" (Excel 2003) or "**PERSONAL.XLSB**" Excel 2007/10. This
 [A]: subroutine has one parameter and is stored in "Module1".
 [A]: The code from above "Call Module1.my_graph_1(my_sheet)" calls this macro from within the
 [A]: "Select Case" statement. The value of "my_sheet" is the name of the EXCEL sheet to process.

```
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 were adjusted
'
```

[B]: This selects a range of cell to work with, in this case it is the output from the SAS program and the range is
 [B]: "Hard-Coded" to be exactly the data that was output by the PROC PRINT command

```
Range("A1:B11").Select
```

[C]: This EXCEL Command adds a chart to the workbook

```
Charts.Add
```

[D]: This command declares the chart will be a pie chart

```
ActiveChart.ChartType = xl3DPie
```

[E]: Here the data source (sheet and range) is defined and how the plot will be oriented (graphed by columns)

```
ActiveChart.SetSourceData Source:=Sheets(my_sheet).Range("A1:B11"), _
PlotBy :=xlColumns
```

[F]: Here the location of the graph is defined (as an object on this sheet, not a separate sheet) and

```
ActiveChart.Location Where:=xlLocationAsObject, Name:=my_sheet
```

[F]: The title of the chart is defined here (yes it has a title, and the text contains)

```
With ActiveChart
.HasTitle = True
.ChartTitle.Characters.Text = "My_new_Pie_chart"
End With
```

[G]: Declare the chart will have a legend

```
ActiveChart.HasLegend = True
```

[H]: Put the legend at the bottom of the graph

```
ActiveChart.Legend.Select
Selection.Position = xlBottom
```

[I]: Allow the fonts on the legend to change size if the graph is resized by the user

```
ActiveChart.Legend.Select
Selection.AutoScaleFont = True
```

[J]: In step [I]: the "ActiveChart.Legend" was selected to be the current object to process. The following EXCEL code construct has no real equivalent in SAS. The "Select With" and "End With" boundaries define a block of code that was previously "Selected" as the active object to work with. Therefore this EXCEL Code segment is modifying the "ActiveChart.Legend" and further qualifying the code to act upon only the "FONT"

[J]: The selections made by the code are relatively easy to guess

```
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
```

[K]: This code positions the "ActiveChart.Legend" to the left of the graph

```
Selection.Position = xlLeft
```

[L]: This command selects the chart for further refinement.

```
ActiveChart.ChartArea.Select
```

[M]: This code is defining aspects of how the chart will be viewed.

```
With ActiveChart
    .Elevation = 35
    .Perspective = 30
    .Rotation = 0
    .RightAngleAxes = False
    .HeightPercent = 100
End With
```

[N]: This code was modified to make the macro more generic.

```
' 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
```

[O]: This code turns off the "Task Pane" command bar that would allow the user to modify the graph in real time with the mouse. Right clicking on the graph will display a menu.

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

[A]: This is the end of the macro. (EXCEL subroutine)

```
End Sub
```