SESUG Paper 283-2019

Use Callable VBS and VBA Code Files to Extend the Power of SAS® to Format Microsoft Excel Worksheets

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

ABSTRACT

Did you ever wish you could use the power of SAS® to take control of Microsoft Excel and make Excel do what you wanted when you wanted? Well, one letter is the key to doing just that, the letter "X", as in the SAS "X" command that opens the door to all operating system commands from SAS. The Microsoft Windows operating system comes with a facility to write a series of commands called scripts. These scripts have the ability to open and reach into the internals of Excel. Scripts can load, execute, and remove Visual Basic for Applications (VBA) macro code and control Excel. This level of control enables you to make Excel do what you want, without leaving any traces of a macro behind. This is Power.

INTRODUCTION

Operating systems need to communicate with the running programs and have methods of passing data between the different software packages. On Windows one of those methods of passing and controlling data movement between computer systems (or the operator and a computer software package) is called Visual Basic Scripting (VBS). SAS can communicate with this program through the "X" command. The Visual Basic scripting capability of the Windows Operating systems is very powerful and is included in the Microsoft operating system software. It can open, manipulate, control, and close an Excel program (along with other Microsoft products). This power permits Excel macros to be stored as individual VBA code modules (*.bas) so they can be used by any team member on a project. The ability to call these macros using VBS, allows the creation a standardized set of macros in a library for report formatting. Community level routines can be stored in a macro library for departmental use, while report specific macros can be stored in separate *.bas code directories. Many Excel formats are not available in SAS, while it is possible to generate code with the SAS PROC TEMPLATE procedure; this is not for the faint at heart. Programmers who are unfamiliar with PROC TEMPLATE may find the code difficult to update or write.

This paper will explain a method of building a system of directories, *.BAS files and *.VBS code to execute Excel macros by calling VBS from SAS to format Excel Reports. A basic working knowledge of how to build Excel Visual Basic (VBA) macros is assumed. The rest of the concept will be sketched out here to allow you to build and expand upon your project needs. The process described here stores the Excel macros in a directory as separate code modules that are only used while the macro is running to process the workbook sheets. These macros can be stored in a read-only directory with limited update access that will make them more secure and allow a wider access to the report processing when new data is available. These VBA macros will allow you to take control of EXCEL as a computer programmer and make it do your bidding. The process is simple can be setup using the following general guidelines. This paper has been upgraded to include some of the features of the SAS ODS Excel Destination available with SAS 9.4 M3.

GUIDELINES FOR BUILDING AND USING A VBA MACRO LIBRARY

1. Establish a disk directory where the VBA and VBS code modules can be stored. If multiple users need access it works best to have it available somewhere where everyone can read the directory, like a server location.

2. Publish a standard set of parameters that the VBS routine users can use to access the VBA code modules.
3. Commonly used routines can be stored in a VBA code module that everyone can use.
4. Unique code modules for individual reports can be placed into the directory and called from the VBS script using parameters setup by the SAS routine.
5. SAS can use the "X" command to start the VBS routine and assign the parameter values to process the report. (Note the SAS "X" command is not available to the SAS Enterprise Guide users due to environment restrictions that prevent SAS Enterprise Guide from knowing where it is running).
6. SAS can process the data files to be used as input to the VBS control module. The output from SAS can be any format that the VBA code modules can read.

The SAS "X" command is a powerful command tool. It allows nearly any Operating System command to be executed by starting it from within the SAS program code. Directories can be listed, other SAS jobs can be started, Excel workbooks can be opened and files can be deleted, now we will discuss how a built-in operating system function can be used to control a Microsoft product like EXCEL (and Word). The VBS scripting language is similar to the VBA code language; but, it does have a few minor differences. You can execute the commands stored in a VBS code module (any_file_name.vbs) simply by double clicking on the filename or using the SAS "X" command to run the script. VBS scripts will also accept parameters at invocation, allowing you to control how they work internally. VBA macros can be built by recording the macro from within the EXCEL workbook. These recorded macros generally have a lot of default commands that are not required for a final macro and they also have a lot of workbook and worksheet specific cell references that may be too specific for a general purpose macro.

SAS has the feature of being a top-down programming language, what that means is that code has to be defined before it can be used. Code is presented to SAS as a text stream, and each macro, data step and procedure call must be executed one group at a time. Code at the bottom of the text file is not executed until the step it is contained within is executed, usually at end of the job. However, Object Oriented programming languages like VBA read in the whole set of code routines, compile them, and then passes control of the program to a routine that waits for something to happen. Like a user moves a mouse pointer, clicks on a menu, or pushes a key on the keyboard. Each object has its own list of things you can make it do or do to it. These are beyond the scope of this paper, but some simple things will be explored to show how to start building a set of your own VBA macros to use to create your reports.

Virtually anything that you can do using Excel you can do using VBA, after all Excel was written using the VBA language. So, you can modify cells by adding data to them, outlining them, moving them, copying them, or clearing them. You can add or delete worksheets, and manipulate rows or columns of data. By being able to add or delete worksheets you have the ability to pass information to Excel that can be used in the formatting process and then be able to delete that information. This is something that a TAGSET or the ODS Excel Destination cannot perform. Files can be read, written or converted from one format to another. By writing the VBA code yourself you have control of the order of the actions that Excel takes. The intent here is to show you how to create formatted Excel output files that are ready for delivery to your user in minutes instead of spending much longer doing it yourself. Making them high quality will be left to you. The initial setup of the first run of each report may take a little longer because you need to write VBA code, but every time you execute the report you will save time running and getting it ready to print.

The SAS code listed below in the appendices under the heading "**SAS Code to Create Unformatted Output Excel Workbooks**" is a simple example. It starts with the SASHELP.CLASS file and sorts it by sex and height, then writes two output Excel workbooks. One workbook is a *.xml file using the ODS TAGSET.EXCELXP (extension .xls). The other workbook is written by the SAS ODS Excel Destination. We will also suppress the column with the observation number to make the file a little cleaner. Additionally, the SAS code labeled "**SAS Code to Call VBS/VBA Routines to Format Excel Workbooks**" sets up SAS macro variables to define the following:

Using the SAS TAGSET.EXCELXP:

Variable	Example	Description
Vbs_code	C:\My_VBA_macros\VBS_Execute_script.vbs	VBA subroutine name to execute
Input_Excel	C:\MY_Excel_Files\my_sorted_class_data.xls	Full File path and Input file name
Output_Excel	C:\MY_Excel_Files\my_sorted_class_and_graph_data.xlsx	Full File path and Output file name
Bas_Code_Path	C:\My_VBA_macros\	VBA Path
VBA_Module	Class_Graph	VBA module name (without the bas)
VBA_Code	Class_Graph	VBA subroutine name to execute

Table 1. List of Parameters used to control the execution of the VBS script shown below.

Using the SAS ODS EXCEL DESTINATION:

Variable	Example	Description
Vbs_code	C:\My_VBA_macros\VBS_Execute_script.vbs	VBA subroutine name to execute
Input_Excel	C:\MY_Excel_Files\sorted_class_xlsx.xlsx	Full File path and Input file name
Output_Excel	C:\MY_Excel_Files\sorted_class_n_graph_xlsx.xlsx	Full File path and Output file name
Bas_Code_Path	C:\My_VBA_macros\	VBA Path

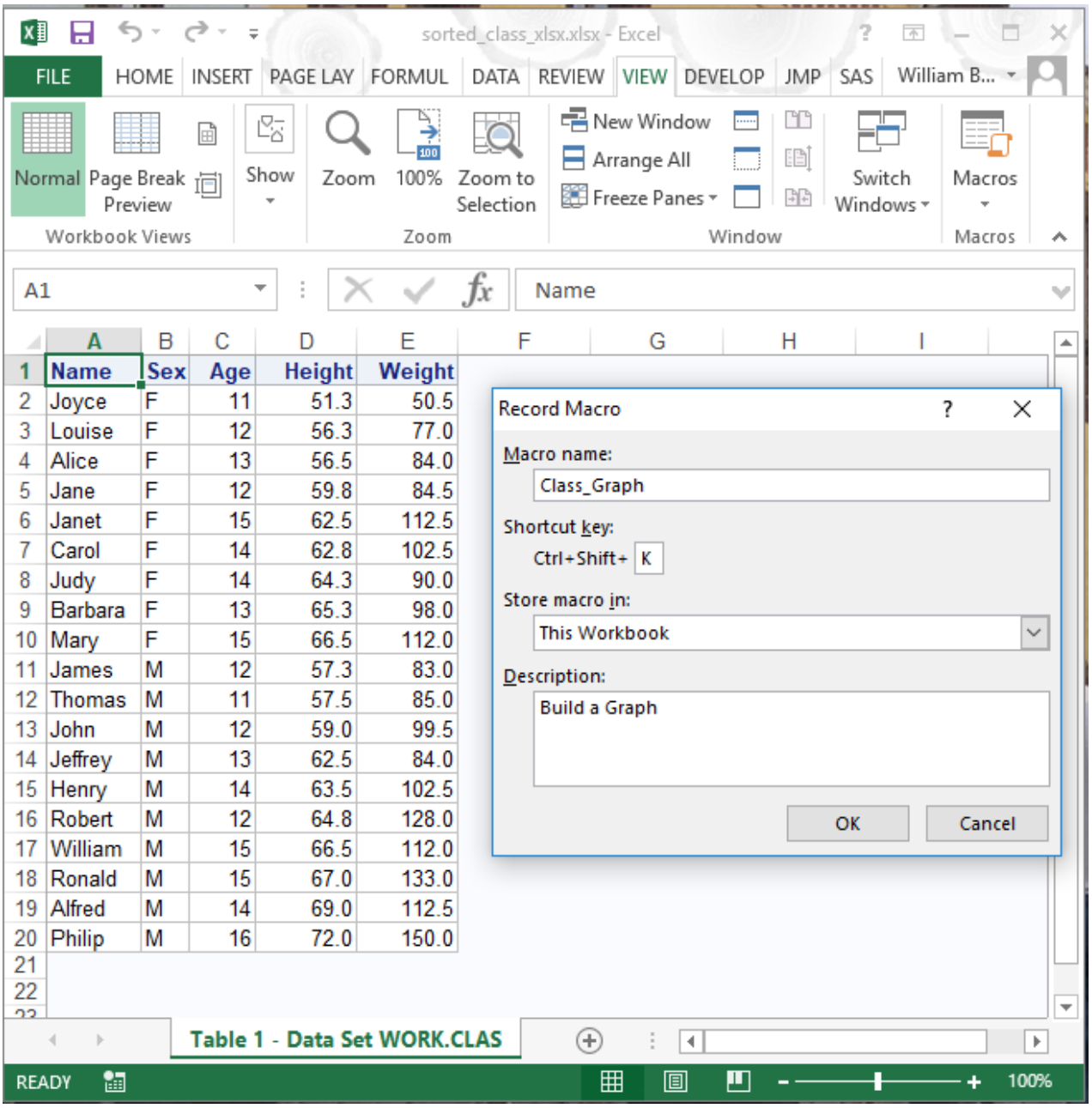
VBA_Module	Class_Graph	VBA module name (without the bas)
VBA_Code	Class_Graph	VBA subroutine name to execute

Table 2. List of Parameters used to control the execution of the VBS script shown below.

The final line of each step of the SAS code is the X command to run the *.vbs script to format the output *.xml file and *.xlsx workbook to create the requested *.xlsx workbooks. It appears twice in the SAS code labeled "**SAS Code to Call VBS/VBA Routines to Format Excel Workbooks**". In most cases you will find SAS documentation that says use the NOXSYNC and the NOXWAIT SAS options. These release the command window and allow SAS to continue running. I suggest when running SAS in a batch or background mode to use XSYNC and the XWAIT (usually the defaults) to help make sure that SAS does not finish running until the Excel formatting has finished. The VBS script code is listed below under the heading "**VBS Script to process a VBA macro**", and the SAS X command is shown here. The single quotes, double quotes, and spaces are important, do not miss any, there are also no commas.

```
X "'&VBS_code.' ""&Input_Excel"" ""&output_excel"" ""&bas_code_path"" ""&vba_module""
""&vba_code"" ";
```

Next, we will create a macro to do some work and save it to a disk file as a VBA code module. Here we have selected the "View" tab on the Excel ribbon and will start to record an Excel macro. We can call the macro anything but here we'll call it Class_Graph. At this point we are developing the VBA code to do the real work of this project. Since this work is done during code development time we will save the Class_Graph VBA macro into "This Workbook", but we will delete it later when we save it to a disk *.bas file. Now we will build a graph to display the sorted data from the Excel worksheet we generated. I noticed a difference between the output default sheet names generated by EXCEPXP and the ODS EXCEL output routines, since my VBA macro references a specific sheet name this caused a problem. My VBA macro was looking for the name 'Table 1 - Data Set WORK.CLASS' which has 29 characters. However the ODS Excel routine only output 28 characters for the sheet name and output the default sheet name 'Print 1 - Data Set WORK.CLAS'. This made the name look different in each output Excel workbook. Because I recorded the VBA macro it gave me a fully qualified sheet and range. So rather than try to figure out how to convert that to a VBA relative address I choose to fix the problem by using SAS to supply a consistent sheet name in the Excel workbooks then I adjusted the VBA macro to match the sheet names. See the SHEET_NAME parameter in the SAS code. For this project both the TAGSET.EXCELXP and the ODS EXCEL Destination used the sheet name **'Table 1 - Data Set WORK.CLAS'.**



	A	B	C	D	E
1	Name	Sex	Age	Height	Weight
2	Joyce	F	11	51.3	50.5
3	Louise	F	12	56.3	77.0
4	Alice	F	13	56.5	84.0
5	Jane	F	12	59.8	84.5
6	Janet	F	15	62.5	112.5
7	Carol	F	14	62.8	102.5
8	Judy	F	14	64.3	90.0
9	Barbara	F	13	65.3	98.0
10	Mary	F	15	66.5	112.0
11	James	M	12	57.3	83.0
12	Thomas	M	11	57.5	85.0
13	John	M	12	59.0	99.5
14	Jeffrey	M	13	62.5	84.0
15	Henry	M	14	63.5	102.5
16	Robert	M	12	64.8	128.0
17	William	M	15	66.5	112.0
18	Ronald	M	15	67.0	133.0
19	Alfred	M	14	69.0	112.5
20	Philip	M	16	72.0	150.0
21					
22					

Figure 1 Excel worksheet with the SASHELP Class data shown and starting to record a VBA macro.

The VBA macro that was recorded using the name, sex, age, height, and weight columns. After the graph was built it was moved near the data table and enlarged to fill the rest of the screen, as in Figure 2 below. The key strokes that you use to build your macro may result is slightly different macro code than what is depicted here.

NOTE – The macro name in the EXCEL Properties window shows Class_Graph in a VBA module called "**Module1**". We can change that by typing a new name in the Properties Window for the module, we will also call it "Class_Graph".

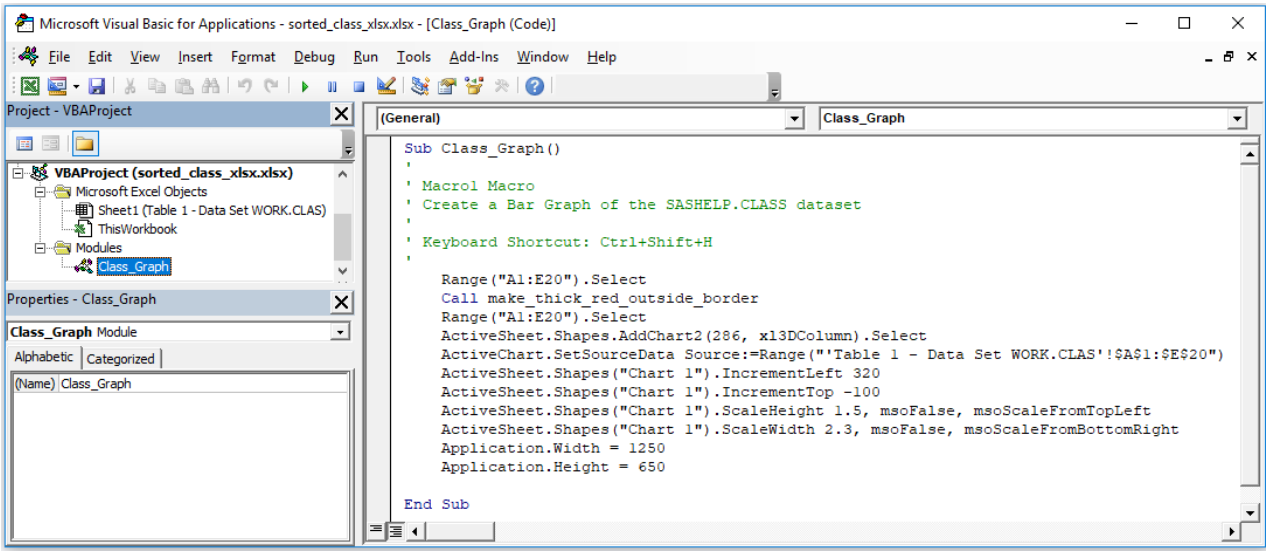
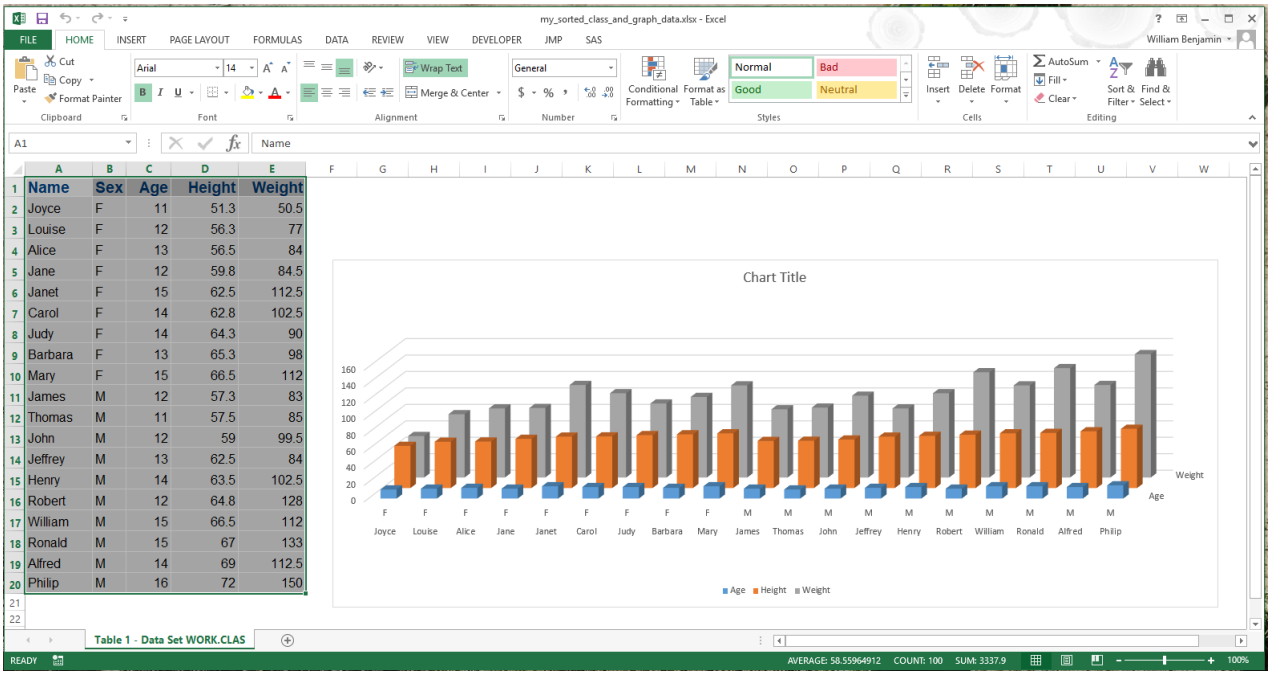


Figure 2 Excel macro code to highlight and build a graph.

Figure 2 contains VBA code to select the cell range A1 to E20 and place a thick red boarder around the cells. This is done with the VBA user defined macro "make_thick_red_outside_border" which will be described later. When control returns to this macro (Class_Graph) the range is selected again and a simple 3D bar chart is created. Note the data source range is hard coded to exist on the worksheet created by SAS with the SHEET_NAME parameter in the two SAS Code routines.



	Name	Sex	Age	Height	Weight
2	Joyce	F	11	51.3	50.5
3	Louise	F	12	56.3	77
4	Alice	F	13	56.5	84
5	Jane	F	12	59.8	84.5
6	Janet	F	15	62.5	112.5
7	Carol	F	14	62.8	102.5
8	Judy	F	14	64.3	90
9	Barbara	F	13	65.3	98
10	Mary	F	15	66.5	112
11	James	M	12	57.3	83
12	Thomas	M	11	57.5	85
13	John	M	12	59	99.5
14	Jeffrey	M	13	62.5	84
15	Henry	M	14	63.5	102.5
16	Robert	M	12	64.8	128
17	William	M	15	66.5	112
18	Ronald	M	15	67	133
19	Alfred	M	14	69	112.5
20	Philip	M	16	72	150

Figure 3. The sample graph that was built while recording a macro.

The intent was to have a macro always build a graph similar to Figure 3. As mentioned above minor changes to the macro may be required to make them work in a generic fashion. In this case the specific reference to "Chart 1" may not always produce a graph; sometimes it will produce an error message and send you to the Excel Debugger. You may see a message similar to the following (Figure 4). This may occur if there are more than one Chart created

in the Excel worksheet. If this graph had been created using SAS Graph the output graph in EXCEL would be a static graph. However, using this method the resulting graph is a native Excel Graph. Because it is a native EXCEL graph, it inherits the EXCEL "ACTIVE-X" features. See Figure 5.

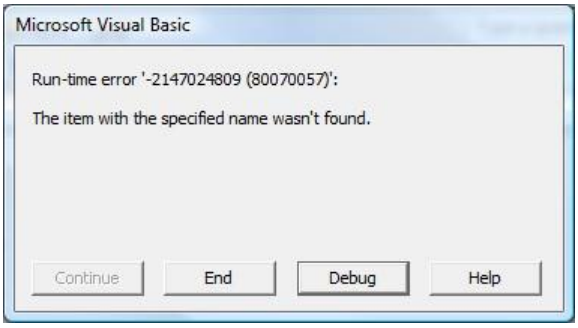


Figure 4 Error message because "Chart 1" may not exist when the macro runs. A second execution may produce "Chart 2".

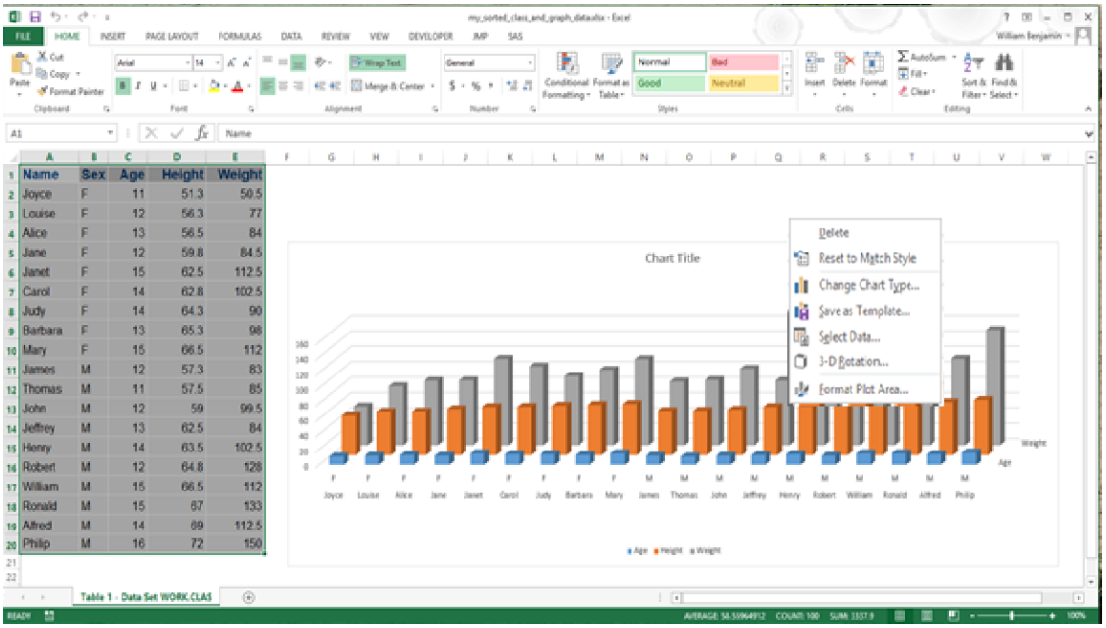


Figure 5 – a "Right Click" on the Chart will show one of several "ActiveX" menus that will allow you to modify the chart attributes.

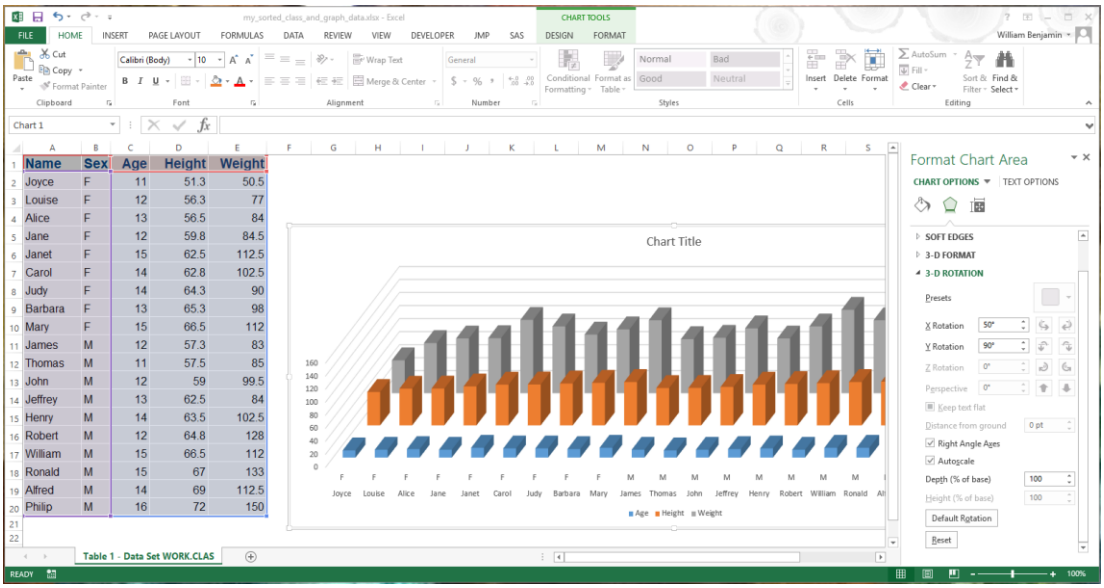


Figure 6. Showing the 3D Selection menu. This has allowed me to change the presentation of the graph.

Now there is one last thing you may want to do before we export this Excel VBA code into a macro library for later use. You can change the name of the macro to something more meaningful for the future use of the macro. I did not do that for this project. This can be done by editing the name property on the left side to the Visual Basic for Applications window, as shown in Figure 2, but I left it alone as CLASS_GRAPH.

The last step here in Excel will be to record another macro, called "**make_thick_red_outside_border**" Figure 7 below has been optimized by removing default commands, consolidating instructions and removing the selection of specific cells to process. These steps make this routine reusable by calling it after selecting a range of cells to process. Additionally, the module name is changed to Common. While this VBA code module only has one VBA subroutine it could contain many subroutines. (NOTE - all VBA modules can contain many subroutines) This code is to show you how to establish a pattern of how to build these modules and subroutines for generic use. If you can do it for one module you can do it for many.

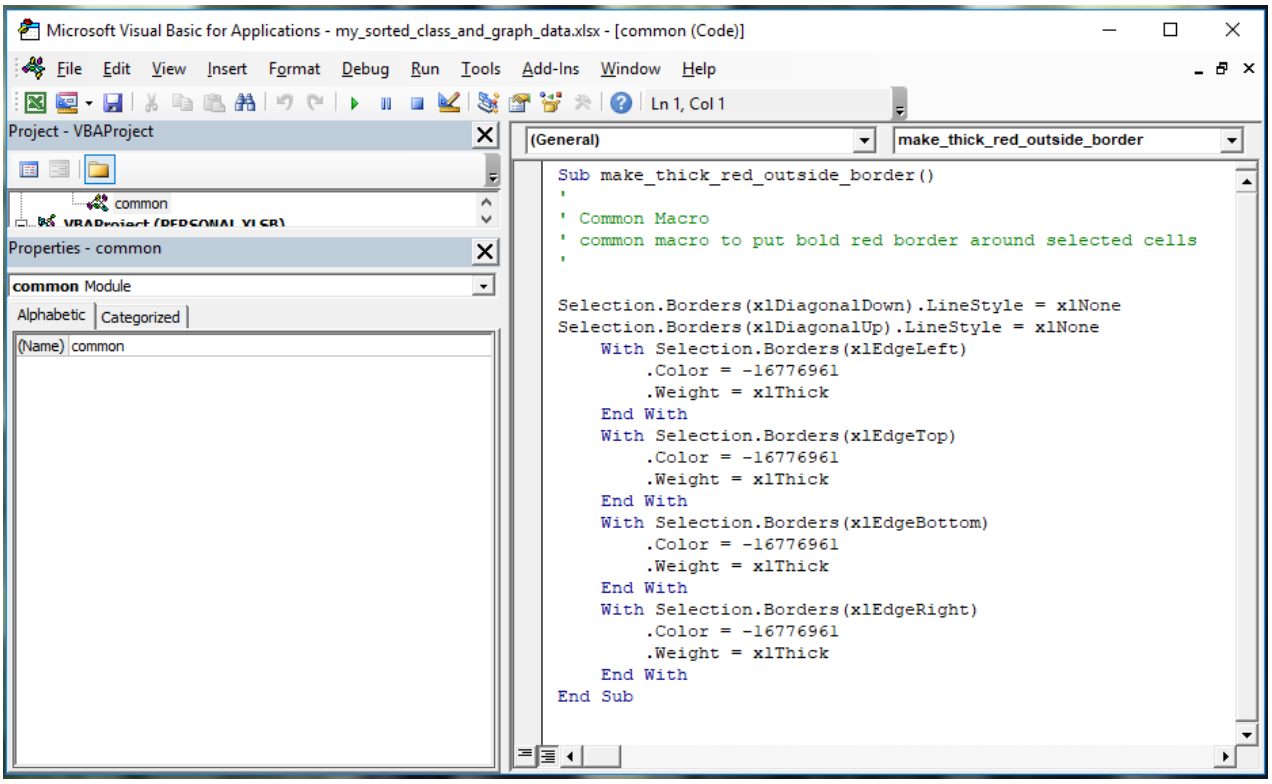


Figure 7 Macro to put a thick red border around selected cells.

Now that we have a main routine to build our graph and a common routine that will put a red border around selected cells we should write some code to use both pieces of code. The simple way to execute the subroutine to make red borders is to call it by name as in the following command:

"Call make_thick_red_outside_border"

All we need to do is insert this command into our original code as indicated in Figure 2.

By selecting the **FILE > EXPORT FILE** options on the VBA Desk Top screen you can "Export" a VBA code module to any disk file you can access. The "Export File" window opens and allows you to browse you system for a location to store the output *,vbs code modules.

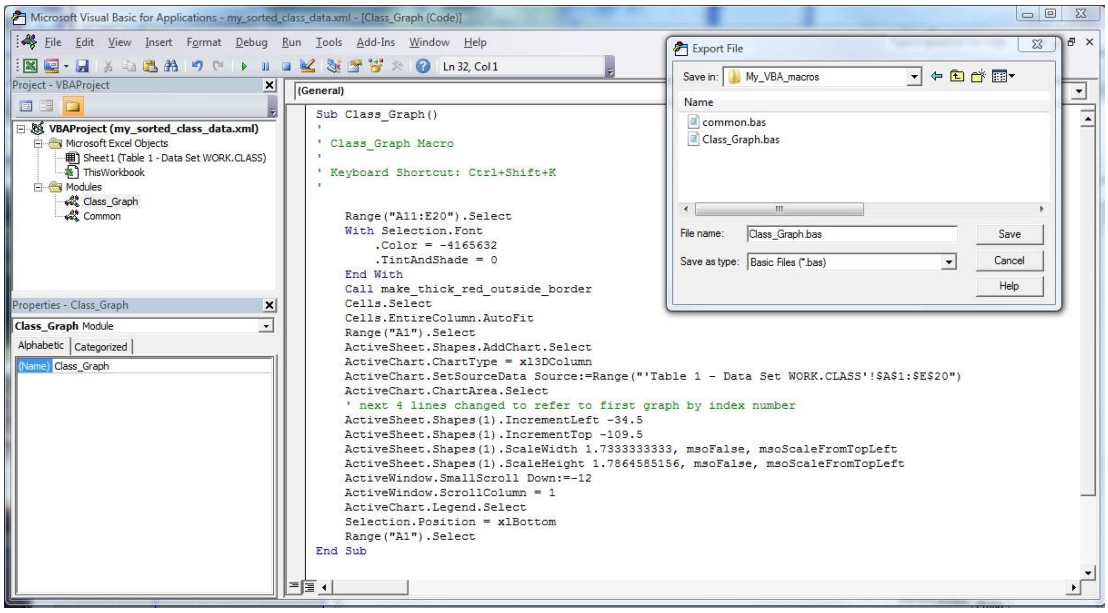


Figure 8. Saving the Class_Graph macro to the My_VBA_macros directory.

Now for the fun part, we need a VBS routine to do the work. We need it to open the Excel Workbooks we created, load our VBA macros, build our graph, and save our data in the XLSX format. When the SAS Code uses the same name for the Input_Excel workbook as the Output_Excel workbook the workbook format is not changed. The code is reproduced later in the appendices as "**VBS Script to process a VBA macro**", and has comments on to describe the action taken by the code. Space here does not permit a line by line description here of the code and how it works.

Another message that may appear could be something like the one in Figure 9. That talks about trusting the Visual Basic Project code. This message will be slightly different depending upon the version of Excel you are using. Here in Figures 9, 10, and 11 are messages that commonly occur when Excel 2010 is being used. The messages are similar for Excel 2013 and Excel 2016.

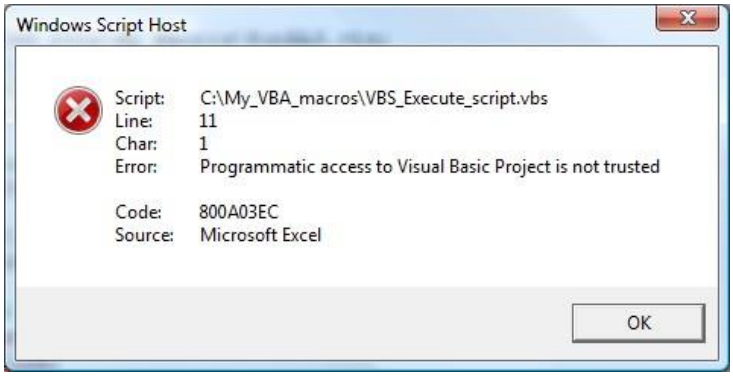


Figure 9 The Windows message about the Visual Basic Project not being trusted.

You may be able to eliminate this message by changing the Trust center options for your computer. If your System Administrator has restricted these commands, then you will need to ask them for help.

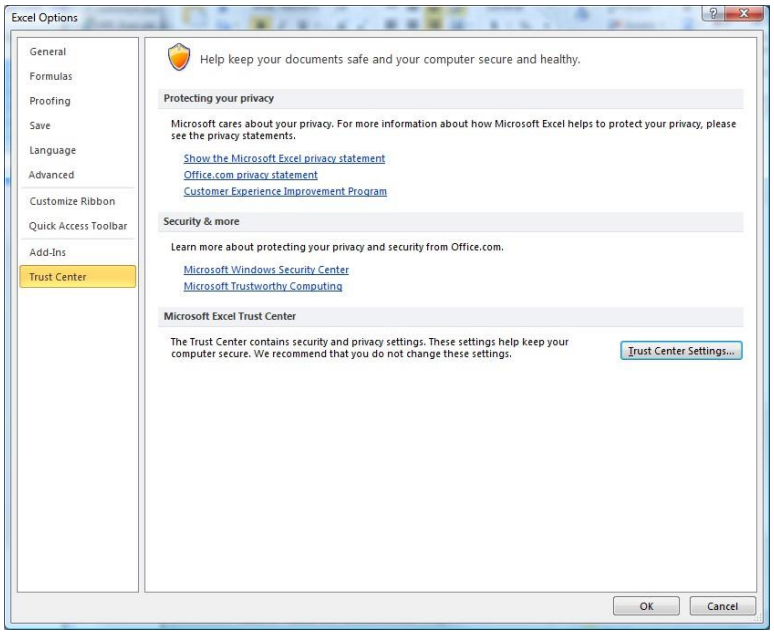


Figure 10 Excel Options window showing the "Trust Center Settings" screen.

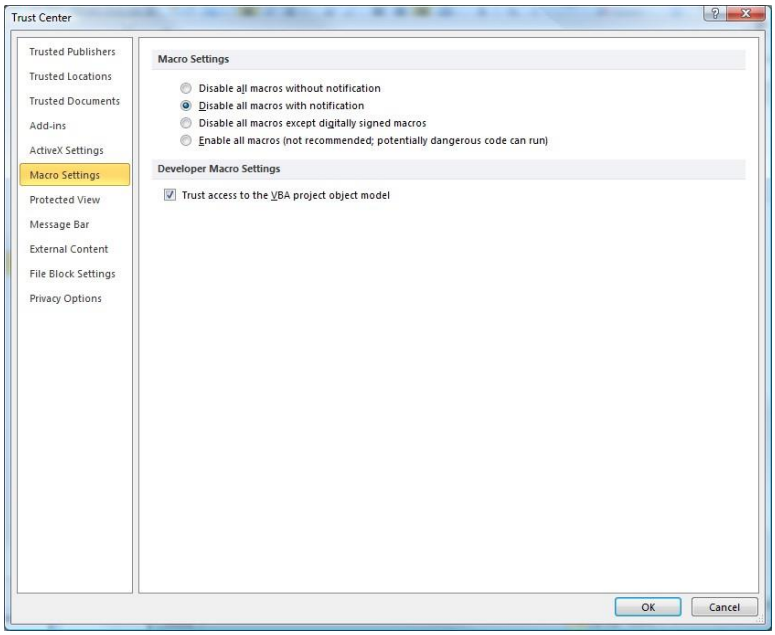


Figure 11 The Excel 2010 "Trust Center Settings" screen showing settings to enable the Trust access to the VBA Project object model checkbox.

It is always wise to verify if you are permitted to modify these settings, Some companies have strict policies about modifying PC Settings without permission.

CONCLUSION

The ability to control how Excel formats a spreadsheet or what data or graphs are placed onto a spreadsheet is a powerful extension of skill. Then add to the process the fact that there are no VBA macros left in the Excel file to make the security tools stop the "Spread" of the file and you now have a powerful tool that will repeatedly save you time. This paper was originally presented in 2013, so it was written before either of my books were published. I was working at a company and developed similar code for a tool they used there. When my first book [1] was published Chapters 12, 13, and 14 were written to demonstrate how to develop a tool this powerful on a simple budget using the SAS TAGSET.EXCELXP to create

the input Excel Workbooks. What this paper describes most closely resembles the material found in Chapter 13 [1] of my first book. A working version of the Chapter 14 tool can be downloaded from the SAS web site https://www.sas.com/store/search.ep?storeCode=SAS_US&keyWords=benjamin&submit=Search

When the SAS ODS EXCEL Destination was released I began writing my second book [2] to describe the options and features of the SAS ODS EXCEL Destination. Because the VBS script and VBA code uses an Excel workbook as input, either of these SAS tools can produce a workbook that can be used as the source the tool described in this paper.

As a side note, after nearly 8 years working on my first book, in the final review three months before the book was published someone at SAS mentioned that the last chapter of the book looked similar to a SAS Product they sold. My response was yes it may look a lot like stored processing but it is the "Cap-Stone" of the book. Please allow it to remain.

REFERENCES

[1] Benjamin, William E., Jr. 2015. Exchanging Data Between SAS® and Microsoft Excel: Tips and Techniques to Transfer and Manage Data More Efficiently.

Cary, NC: SAS Institute Inc.

[2] Benjamin, William E., Jr. 2017. *Exchanging Data* From SAS® to Excel: The ODS Excel Destination. Cary, NC: SAS Institute Inc.

[3] SAS Institute Inc. 2016. SAS® 9.4 Output Delivery System: User's Guide, Fifth Edition. Cary, NC: SAS Institute Inc.

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: 623-337-0269

E-mail: wmebenjaminjr3@juno.com

APPENDICES

SAS CODE TO CREATE UNFORMATTED OUTPUT EXCEL WORKBOOKS

```
/* using Excel 2013 at "C:\Program Files\Microsoft Office 15\root\office15\excel.exe"  */
/* using SAS Version 9.4 TS Level 1M3                                                 */

options symbolgen mprint mprintnest mlogic mlogicnest linesize=150;

PROC SORT data=sashelp.class out=class;
BY sex Height;
RUN;

/*****************************************************************************************/
/* When using the TAGSET.EXCELXP code, if outputting a workbook as a .XML see the following:  */
/* ending the output file name in .XML may cause formatting errors and cause the job to fail  */
/*                                                                                            */
/* ending the file name in .XLS may result in a message from Excel, but the file will open    */
/* the message may be something like - The file format and extension of 'your file name.xls'  */
/* don't match. The file could be corrupted or unsafe. Unless you trust its source don't open */
/* it. Do you want to proceed anyway?                                                         */
/*                                                                                            */
/*****************************************************************************************/
/* I noted a difference between the output default sheet names generated by EXCELXP and the   */
/* ODS EXCEL output routines, Since my VBA macro references a specific sheet name this caused */
/* a problem. My VBA macro was looking for the name 'Table 1 - Data Set WORK.CLASS' which has */
/* 29 characters. However the ODS Excel routine only output 28 characters for the sheet name  */
/* and output the default sheet name 'Print 1 - Data Set WORK.CLAS'. This made the name look   */
/* different in each output Excel workbook. Because I recorded the VBA macro it gave me a     */
/* fully qualified sheet and range. So rather than try to figure out how to convert that to a */
/* VBA relative address I choose to fix the problem by using SAS to supply a consistent sheet */
/* name in the Excel workbooks. and then adjust the VBA macro to match the sheet names.       */
/*                                                                                            */
/*****************************************************************************************/
/* The ODS "ID" option allows me to write two outputs with one code process                   */
/* By default neither EXCELXP Tagset or ODS EXCEL minimizes column widths, but the            */
/* ABSOLUTE_COLUMN_WIDTH option will, I did this to get the data to fit clearly on one page.  */
/*****************************************************************************************/

ODS tagsets.ExcelXP (id=EXCELXP)
          file="C:\MY_Excel_Files\my_sorted_class_data.xls"
          options(sheet_name='Table 1 - Data Set WORK.CLAS'  /* sheet name    */
                  ABSOLUTE_COLUMN_WIDTH='6,3,4,6,6'         /* column widths in charcters   */);
```

```
ODS EXCEL           (id=ODSEXCEL)
          file="C:\MY_Excel_Files\sorted_class_xlsx.xlsx"
          Options(sheet_name='Table 1 - Data Set WORK.CLAS'  /* sheet name    */
                  ABSOLUTE_COLUMN_WIDTH='7.5,3.5,5,8,8'      /* column widths */);

PROC PRINT data=class noobs;
RUN;

ODS tagsets.ExcelXP (id=EXCELXP)  Close;
ODS Excel           (id=ODSEXCEL) Close;
```

SAS CODE TO CALL VBS/VBA ROUTINES TO FORMAT EXCEL WORKBOOKS

```
options noquotelenmax; * turn off long quoted string error messages;

* the SAS "X" command "NOWAIT" option is turned off so the SAS sode waits until the Excel job finishes;


/*  SAS Code to Create an Unformatted Output EXCELXP (XML) *.XLS File - see note above */


%let vbs_code      = C:\My_VBA_macros\VBS_Execute_script.vbs;              * VBS subroutine name to execute ;

%let Input_Excel   = C:\MY_Excel_Files\my_sorted_class_data.xls;           * Full path and Input file name  ;

%let output_excel  = C:\MY_Excel_Files\my_sorted_class_and_graph_data.xlsx; * Full path and Output file name ;

%let bas_code_path = C:\My_VBA_macros\;                                   * Full path Location of bas file ;

%let vba_module    = Class_Graph;                                         * VBA Path and module name (without
the bas);

%let vba_code      = Class_Graph;                                         * VBA subroutine name to execute ;

X "'&VBS_code.' ""&Input_Excel"" ""&output_excel"" ""&bas_code_path"" ""&vba_module"" ""&vba_code"" ";


/*  SAS Code to Create an Unformatted Output *.XLXS File */


%let vbs_code      = C:\My_VBA_macros\VBS_Execute_script.vbs;         * VBS subroutine name to execute ;

%let Input_Excel   = C:\MY_Excel_Files\sorted_class_xlsx.xlsx;        * Full path and Input file name  ;

%let output_excel  = C:\MY_Excel_Files\sorted_class_n_graph_xlsx.xlsx;* Full path and Output file name ;

%let bas_code_path = C:\My_VBA_macros\;                               * Full path Location of bas file ;

%let vba_module    = Class_Graph;                                     * VBA Path and module name (without the
bas);

%let vba_code      = Class_Graph;                                     * VBA subroutine name to execute ;


X "'&VBS_code.' ""&Input_Excel"" ""&output_excel"" ""&bas_code_path"" ""&vba_module"" ""&vba_code"" ";

options quotelenmax; * turn on long quoted string error messages;
```

VBS SCRIPT TO PROCESS A VBA MACRO

```
Dim Input_Excel, output_excel, bas_code_path, vba_module, vba_code, objxl, objwk,
vbCom, myMod

Input_Excel  = WScript.Arguments(0) 'Full File path and Input file name
output_excel = WScript.Arguments(1) 'Full File path and Output file name
bas_code_path= WScript.Arguments(2) 'Full File path Location of .bas file
vba_module   = WScript.Arguments(3) 'VBA module name (without the .bas)
vba_code     = WScript.Arguments(4) 'VBA subroutine name to execute

set objxl = CreateObject("Excel.Application") 'Start Excel
set objwk = objxl.Workbooks.Open(Input_Excel) 'Open the input file

'Activate special software
set vbCom = objxl.ActiveWorkbook.VBProject.VBComponents

 objxl.DisplayAlerts = wdAlertsNone    'no quote to Turn off error messages
                                       'a  quote to turn on  error messages
if vba_module <> "" then
  vbCom.Import ("" & bas_code_path & vba_module & ".bas") 'Import VBA Code
  vbCom.Import ("" & bas_code_path & "Common.bas")  'Import Common Routines

  objxl.Run "" & vba_module & "." & vba_code & ""    'Run VBA Code

  'Remove VBA modules
  Set myMod = objxl.ActiveWorkbook.VBProject.VBComponents("" & vba_module & "")
     objwk.VBProject.VBComponents.Remove myMod

  Set myMod = objxl.ActiveWorkbook.VBProject.VBComponents("Common")
     objwk.VBProject.VBComponents.Remove myMod

end if

'Save as Excel *.xlsx workbook and close input workbook
if Input_Excel <> output_excel then
       objxl.ActiveWorkbook.SaveAs output_excel, 51 'Excel xlsx format
       objxl.Workbooks.Open(output_excel).Close
End if

if Input_Excel = output_excel then objxl.ActiveWorkbook.Save

objxl.Workbooks.Open(input_excel).Close
```

```
objxl.Quit
set objxl = nothing
set objwk = nothing
```

* NOTE – Previous versions of this macro differ in the way the Excel workbooks are closed, as of 07/29/2018 this version is the most current and closes both the input and output Excel workbooks if different names are given.

VBA COMMON MODULE WITH MACRO CODE TO BUILD A THICK RED BORDER

```
Attribute VB_Name = "common"
Sub make_thick_red_outside_border()
'
' Common Macro
' common macro to put bold red border around selected cells
'


Selection.Borders(xlDiagonalDown).LineStyle = xlNone
Selection.Borders(xlDiagonalUp).LineStyle = xlNone
    With Selection.Borders(xlEdgeLeft)
        .Color = -16776961
        .Weight = xlThick
    End With
    With Selection.Borders(xlEdgeTop)
        .Color = -16776961
        .Weight = xlThick
    End With
    With Selection.Borders(xlEdgeBottom)
        .Color = -16776961
        .Weight = xlThick
    End With
    With Selection.Borders(xlEdgeRight)
        .Color = -16776961
        .Weight = xlThick
    End With
End Sub
```

* NOTE – The “Attribute” command on the first line is output by the “Export” process. It is also used by the “Import” process but not part of the VBA code. It is not visible in the Excel VBA desk top images, but must be in the *.bas file.

FINAL VBA MACRO CODE TO FORMAT THE WORKSHEET AND BUILD A GRAPH

```
Attribute VB_Name = "Class_Graph"
Sub Class_Graph()
Attribute Class_Graph.VB_Description = "Create a Bar Graph of the SASHELP.CLASS
dataset"
Attribute Class_Graph.VB_ProcData.VB_Invoke_Func = "H\n14"
'
' Macro1 Macro
' Create a Bar Graph of the SASHELP.CLASS dataset
'
' Keyboard Shortcut: Ctrl+Shift+H
'
    Range("A1:E20").Select
    Call make_thick_red_outside_border
    Range("A1:E20").Select
    ActiveSheet.Shapes.AddChart2(286, xl3DColumn).Select
    ActiveChart.SetSourceData Source:=Range("'Table 1 - Data Set
WORK.CLAS'!$A$1:$E$20")
    ActiveSheet.Shapes("Chart 1").IncrementLeft 320
    ActiveSheet.Shapes("Chart 1").IncrementTop -100
    ActiveSheet.Shapes("Chart 1").ScaleHeight 1.5, msoFalse, msoScaleFromTopLeft
    ActiveSheet.Shapes("Chart 1").ScaleWidth 2.3, msoFalse, msoScaleFromBottomRight
    Application.Width = 1250
    Application.Height = 650

End Sub
```

* NOTE – The “Attribute” command on the first line is output by the “Export” process. It is also used by the “Import” process but not part of the VBA code. It is not visible in the Excel VBA desk top images, but must be in the *.bas file.

** NOTE – The ActiveChart.SetSourceData object uses a hard coded sheet name that matches the SHEET_NAME assigned in the SAS Code to generate the output Excel Workbooks. The Excel Cell ranges of the data are also hard coded as ("A1:E20") and “!$A$1:$E$20”.