Paper HOW-10

Give the Power of SAS® to Excel Users Without Making Them Write SAS Code

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

ABSTRACT

Merging the ability to use SAS® and Microsoft Excel can be challenging. However, with the advent of SAS® Enterprise Guide®, SAS® Integration Technologies, SAS® BI Server software, JMP® software, and SAS® Add-In for Microsoft Office; this process is less cumbersome. Excel has the advantages of being cheap, available, easy to learn, and flexible. On the surface, SAS and Excel seem widely separated without these additional SAS products. But wait, BOTH SAS AND EXCEL CAN INTERFACE WITH THE OPERATING SYSTEM. SAS can run Excel using the command and Excel can run SAS as an "APPLICATION." This is NOT DDE; each system works independently of the other. This paper gives an example of Excel controlling a SAS process and returning data to Excel.

INTRODUCTION

As software interfaces have improved over time the movement and manipulation of SAS and Excel data have become nearly seamless, for the companies that can afford to purchase the products listed in the abstract. In fact you even need to purchase an extra product called SAS/Access for PC files to be able to use a SAS LIBNAME statement to access your Excel files. The process that will be described here will require BASE SAS and Excel 2003 or later. The object of the process is to retrieve data from SAS data files that the end user has access to read. This user will not need to know how to program in either SAS or Excel VBA. But someone at the company will need to be able to do these things to set up the tools for the Excel user. The object is to provide the Excel user with a menu that is available when Excel is started to allow the Excel user to pick a program or report they want to execute. The following is an example of how to give Excel control over SAS processes that return data to Excel.

THE PROCESS

Described in this paper will be an Excel workbook with a Graphical User Interface (GUI) menu that will enable the selection of a process. When the process, which can be nearly any SAS program, is executed by Excel the SAS process will return the output back to Excel. Some SAS tools like the EXCELXP tagset template features of ODS can produce xml files readable by Excel that require only minimal if any formatting. I will also include some instructions that will teach you how to set up and execute Excel VBS/VBA code to execute formatting routines within Excel.

THE STEPS

This paper will describe how to build an Excel GUI with the VBA Commands to Execute SAS code and how to structure your SAS programs to build files that Excel can load and format. In addition to showing you how to pass the data back to Excel, I will also describe some file structures that will enable periodic processing cycles to be established so that the routines can be executed by pointing to new data sources without the need to modify the SAS code for each execution of the program.

THE RESULTS

The length of this paper does not allow for more than a "Bare-Bones" Excel interface with limited features, but all good software projects start that way. This tool will provide a method that will allow the execution of SAS code without the Excel user having the need to read, write, or modify SAS code. Additionally the Reports presented within Excel can be enhanced with print options set and be ready to open and print. This kind of a tool will enable a small programming staff to support an "On-Demand" reporting system.

BUILDING THE TOOL

There are three phases to setting up this tool for use. The first phase is to build the Excel tool that will store the menu and parameters for the Excel part of the process. Building the Menu portion of the Excel tool requires

someknowledge of Excel VBA code and Excel User Forms. Phase two requires building SAS programs that can be executed by the tool and establishing libraries and a naming convention will permit proper execution of the programs. Phase three of the processing only requires that the Excel user has access to the tool and is trained about how to select the programs or reports and what to expect to see while the programs are running.

BUILD AND DISPLAY EXCEL USER FORM

Step by step instructions will not be provided here; however once you know how to get started it is a simple “Chore” to be able to build your own menu. I will show you how to get started and what pieces of the menus I placed into the tool. This will allow you to point and click your way forward to produce your own copy of the tool. To locate the Integrated Development Environment (IDE) first select the worksheet then press the keypad buttons “ALT” and “F11” at the same time. This will open another screen shown below. The screen shot shown below is virtually the same for all versions of Excel that can open the IDE. The VBA code to build the UserForm1 image is not a text based process and cannot be included here.

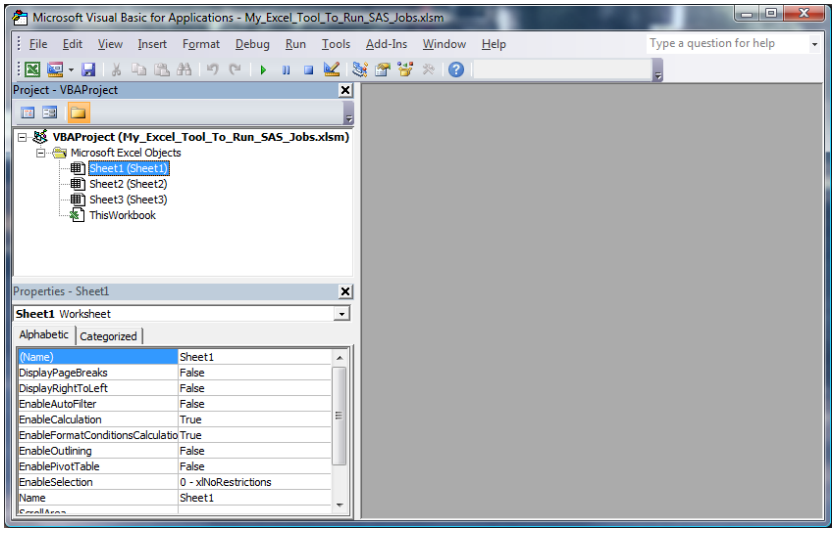

Figure 1 IDE screen with Project and Properties windows visible.

The appendices listed here will enable you to find and arrange the Excel Menu components as shown in the figure in Appendix C.

1. Appendix A contains common terms associated with building Excel GUI menus and Excel tools.
2. Appendix B shows the location of UserForm items/features that can be placed onto the menu.
3. Appendix C pictures the Excel menu in the Excel IDE window.
4. Appendix D describes each item on the tool menu relative to its location on the menu shown in Appendix C.

SET-UP PARAMETERS

Once the UserForm menu is built then it is time to link the properties of the individual menu items to the source fields in the Excel workbook. Normally the lower left hand side of the Excel IDE shows the properties of the highlighted item in the UserForm on the right hand side of the screen. The “Control Source” field on the properties screen indicates what Excel “Cell” will contain the contents that will be linked to the menu control item. The setup used in this example follows in Figure 2 below.

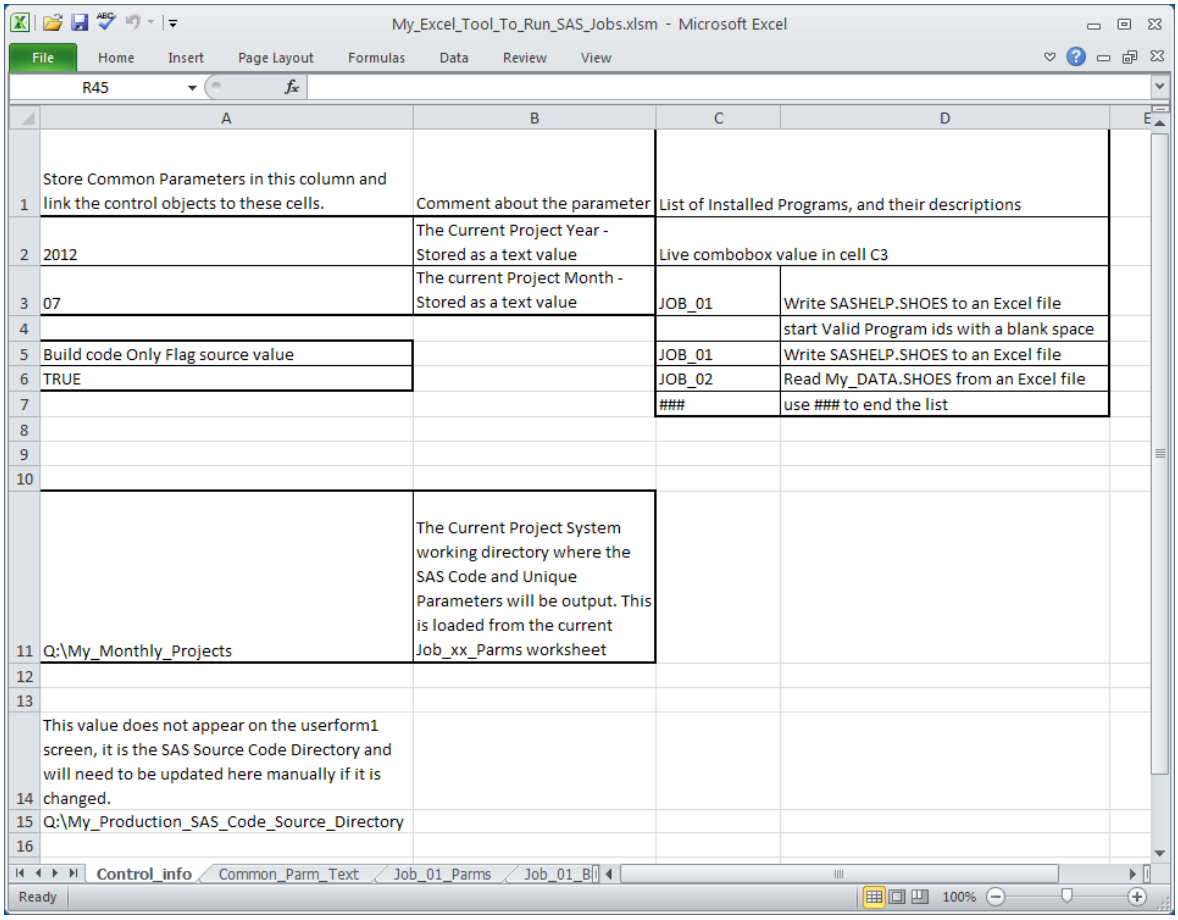

Figure 2. Sample layout of the Control_info worksheet.

For Example TextBox1 which contains the value for variable “Year” is linked to an Excel cell “A2” of sheet “Control_info”. This linkage is set up by setting the TextBox1 ControlSource property field to the value “Control_info!A2” (without the quotes). Similarly, the other fields can be linked to other cells in the worksheet. The “Caption” properties field will hold the value of the Label messages. The values of some of these fields are set at runtime using VBA code.

BUILD PARAMETER STORAGE LOCATIONS WITHIN EXCEL

Another thing that we need is some storage areas within the workbook to store information about the individual programs that the tool will support. For this version of the tool we will need six worksheets in our workbook. Note: The Excel sheet names (Sheet1, Sheet2, etc.) do not need to be in order because we will be using the user-assigned names of the sheets for all of the references to the sheets. The list below describes their usage:

1. Sheet1 – “Control_Info” Storage for tool information like variable values
2. Sheet2 – “Common_Parm_Text” Storage for text output by the tool to control jobs
3. Sheet3 – “Job_01_Parms” Storage for Job 01 Parameters
4. Sheet4 – “Job_01_Backup” Backup Storage for Job 01 Parameters
5. Sheet5 – “Job_02_Parms” Storage for Job 02 Parameters
6. Sheet6 – “Job_02_Backup” Backup Storage for Job 01 Parameters

MAKE USERFORM1 EXECUTABLE BY EXCEL WITH VBA CODE

The VBA code in this figure shows where the Excel VBA code can first take control of a running Excel process. It is important to include this code or the tool will not work!

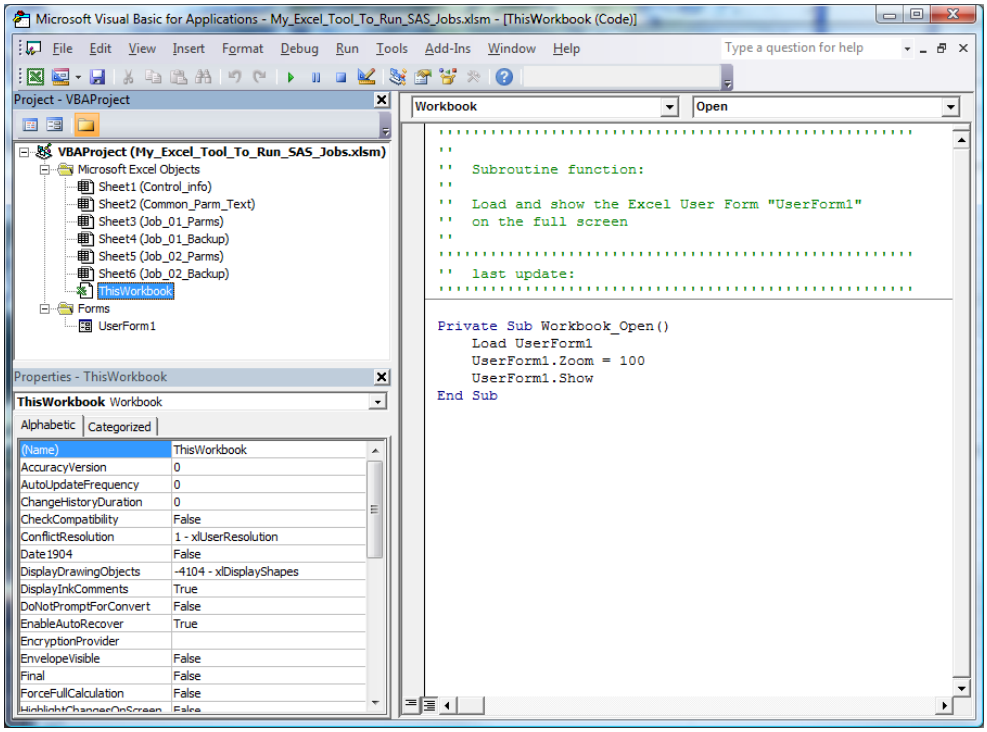

Figure 3. VBA code to display the UserForm menu.

RUN EXCEL IN SAFE-MODE

I recommend that the reporting tool be run in what Microsoft calls the “Safe Mode”. Running the Excel program this way allows the SAS program to also start Excel and send the name of a file for Excel to open. This can be done by putting the following command into a text file (*.txt) and renaming the file to *.bat. Windows will execute a *.bat file by double clicking on the file just like a file named *.exe. The quotes are required because the path to the Excel program will most likely have spaces in the path name. The command is generically **“Path\EXCEL.EXE” /s “file_Path\file_name.xlsm”** the “/s” is a switch indicating that you want to run EXCEL.EXE in “Safe Mode”. Your actual path may vary and the VBA code can be stored in either a “xls”, “xlsx” or “xlsm” file, but not a “xlsb” file.

“C:\Program Files (x86)\Microsoft Office\Office14\EXCEL.EXE” /s “Q:\Path_to_my_Tool\My_tool.xlsm”

CREATE OR MODIFY REPORT PARAMETERS

This tool is designed to store parameters for programs to facilitate the periodic execution of mature reports or processes. I define a mature process or report as software that has predictable consistent inputs and produces reliable consistent outputs. Words like predictable, reliable, and consistent do not mean that every execution of any process or report will produce the same results. It means that the software executes reliably using the current data to produce meaningful outputs. The tool will store parameters that drive the selection of the input data and the criteria for the outputs. The Excel worksheets described above as Sheet3 to Sheet6 are the storage locations for the parameters for the program described by this paper. (the program has a worksheet for the current parameters and a sheet for a set of backup parameters). See the description below. A second set of worksheets are provided for expansion.

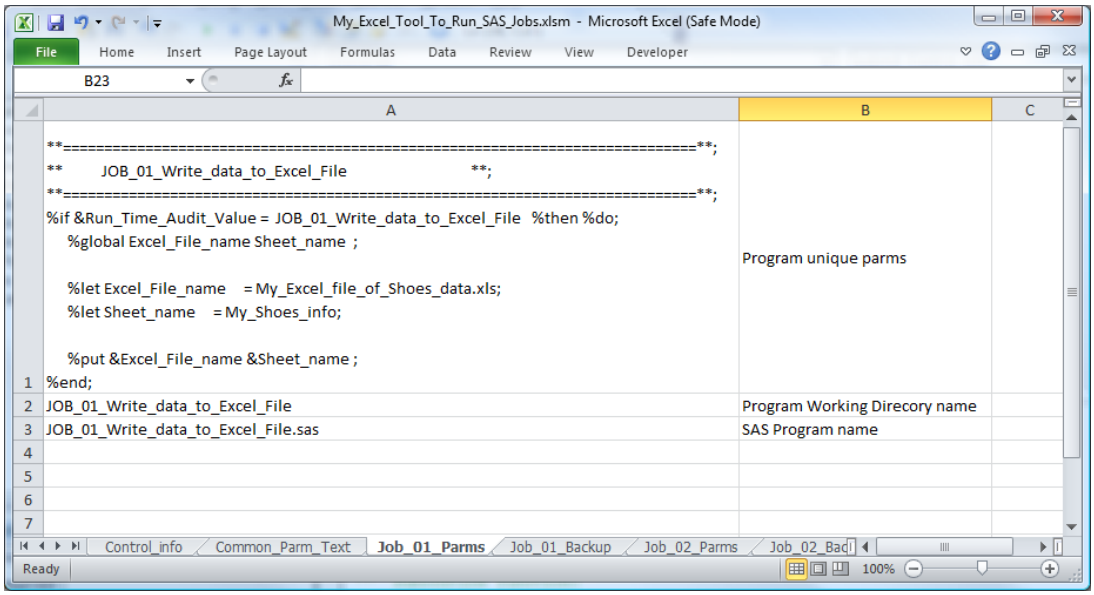

Figure 4. Worksheet to store parameters for executing a SAS program.

HOW VBA CODE WORKS TO RUN SAS AS AN APPLICATION

The VBA Code presented in this paper implements the options shown on the screen in Figure 5 (also in Appendix C in the IDE view). Each element on UserForm1 is described in Appendix D. Including the hidden element between the label “Unique Parameters” and “Build Code Only”. When counting the line with the title “My New Tool To Run SAS Programs” 7 rows of UserForm1 elements are described. The code to implement this tool is in Appendix E. At the time of this writing the power point slide show, code for the tool, and the UserForm1 menu described in this paper are scheduled to be available to be viewed on the SAS Global Forum 2013 web site.

The Tool has the following features:

1. The Year and Month parameters used in file-name paths and are preserved between executions on the tool.
2. The SAS Source program names are stored for each program (This tool has 2 installed)
3. A root work directory is identified for the periodic processing.
4. A drop down menu is provided to enable selection of the program to execute.
5. The “Load Parameters” button populates the “Program Name” on Line two, “Unique Parameters” (the white space on the bottom half of the screen), and a hidden “Prograam_Directory” field on the fifth line.
6. A Checkbox indicates that you wish only to build the code files when this is checked the SAS code is copied to the “Program Directory” but not executed. When left unchecked the tool will execute the SAS code.
7. The two buttons on the lower left of the screen either back-up or restore the program parameters.
8. The “Run Program” button will copy the code to the “Program Directory” and optionally execute the SAS code, based upon the status of the “Build Code Only” checkbox.
9. The “Save and Exit” button does just that. If you wish to exit without saving the changes then click the “X” in the top right of the UserForm1” screen and the workbook.
10. The tool has two control information worksheets (“Common_Parm_Text” and “Control_Info”, each of these can be customized with more features added to the tool.
11. Each SAS Program installed into the tool uses two worksheets. A parameter sheet and a backup parameter sheet.

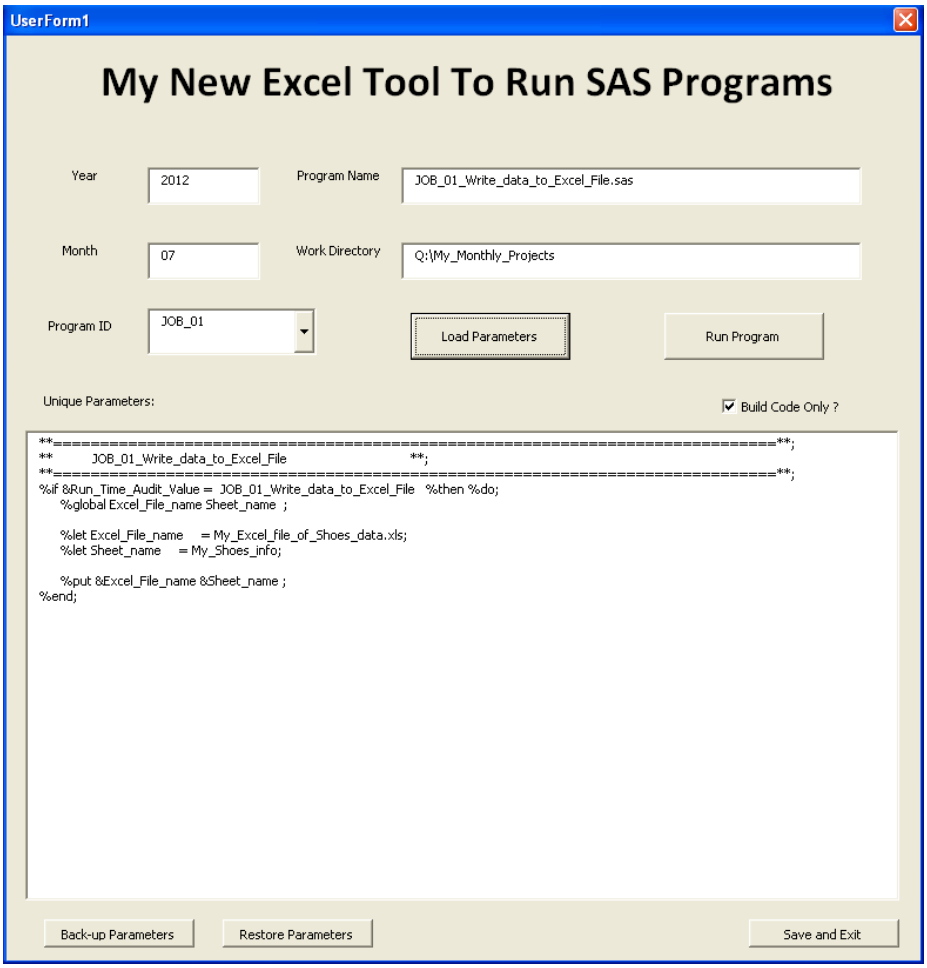

Figure 5. View of Excel Tool with parameters for PGM_01 loaded.

KEY VBA CODE SEGMENTS TO SUBMIT CODE TO SAS AT RUN-TIME

One of the key parts of this tool is the ability of Excel to control other software. About half way down the third page of the code in Appendix E you will find the following code:

```
Set SAS = CreateObject("SAS.Application")
SAS.Visible = True
SAS.Wait = True
```

The above VBA Code in conjunction with the following VBA code showing SAS code commands enclosed in quotes to be executed by SAS give this tool most of its power.

```
'SAS Commands to really include the temp file and run the program
 SAS.submit ("Filename in_src '" + Full_Path + "';")
 SAS.submit ("%include in_src(" + parm_file + ");")
 SAS.submit ("run;")
 SAS.submit ("data _null_; *something at the end to force log output;")
 SAS.submit ("a=1;")
 SAS.submit ("run;")
 SAS.submit ("data _null_;")
 SAS.submit ("completed = setrc('Complete' , 1.0);")
 SAS.submit ("put completed ;")
 SAS.submit ("run;")
```

```
SAS.submit ("proc printto print=print;")
SAS.submit ("data _null_; *something at the end to force log output;")
SAS.submit ("a=1;")
SAS.submit ("run;")
```

EXCEL TOOL FUNCTIONS

Now that we have touched upon the building, structure, features, functions, and key internal parts of the tool, let's look at the overall picture of how it works.

- The tool is designed so that a double click on a desktop icon will start the tool.
- An initialization process builds the drop down menu and loads the parameters onto the menu.
- A program can be selected from a drop down list.
- The parameters for the SAS program can be loaded with the click of a button.
- These parameters can be modified or used as listed.
- Changes to the parameters are saved between sessions.
- Separate users may execute the tool (at different times) saving the latest parameter changes.
- SAS and VBA code is stored in a separate location
- SAS code is copied to a working directory before being used.
- A Batch execution file is created that preserves output log and listing files for auditing.
- Optionally the tool can execute a SAS program directly or allow you to execute the SAS code.
- Excel output worksheets can be configured to allow printing without manual modification.
- Tool usage and maintenance tasks can be distributed to different departments.
- Users of the tool do not need to have any programming skills to run the reports.
- The tool is built with BASE SAS software Version 8 and above.
- The tool is built with off-the-shelf Excel software version 2003 and above.

- The down side is at least one person has to be able to maintain the SAS and Excel/VBA code.

TEACH OTHERS HOW TO USE THE REPORTING TOOL

New users need only to know how to start the program, what the parameters mean, and how to change them. Many programs can be written so that only the quarterly parameters need to be changed. These parameters point to the current period processing files and control the input and output data streams. Of course the scope of what the users can modify is only limited by the SAS programmer's creativity.

RUN-TIME ENVIRONMENT

This process is designed for and executes on a Microsoft Windows operating system. The SAS X command will send commands from BASE SAS to any operating system. Excel typically runs on a Windows operating system, but some Macintosh operating systems have windows emulators that will run Excel. Some UNIX computers can see and retrieve Excel data from a computer running a Windows operating system. But, Visual Basic Scripting (VBS) is native to only windows. This key component is started by the SAS X command and runs a new copy of Excel loading, executing, and deleting VBA macro code from the Excel Workbooks.

The departmental files should conform to some sort of a file naming convention so that the directory levels are predictable. This allows Excel to use values stored to describe data locations reliability. The SAS code can be stored on a secured directory to prevent it from being modified. Since SAS is called by Excel it will be just a little more protected from having the code modified because the user does not have full control of SAS. (Yes as with everything computer related there may be ways around this.)

The tool building process will establish a method of storing data about each SAS program or report and the SAS Data files required to execute the program or report. The Tool Menu will also be available for the Excel user to select the programs or reports and update the parameters required for this instance of running the program or report. The tool will require the inclusion if commands to allow the Excel user to Execute the SAS Program.

After the tool is setup for the Excel users to access, either each user (or group of users) can be given access to the same Excel file with the tool installed, allowing common updating of the parameters. Or each user can have a unique copy of the Excel tool to store their unique report parameters within that file. This gives a net result of an Excel user running an Excel program (menu) to produce SAS derived output files and reports. The output reports are not required to be sent to Excel and the SAS programs can produce any kind of output your programmer can provide.

DESCRIPTION OF SOFTWARE APPENDICES

Appendix E - VBA Code for Excel Workbook Tool to run SAS as an Application

This VBA Code performs the following general tasks:

1. UserForm1 initialization
2. Program selection from a drop down menu
3. Update parameters
4. Load program unique parameters
5. Execute SAS programs
6. Copy SAS programs to working directories
7. Create files of SAS code with program quarterly parameters
8. Create a batch execution file
9. Maintain current and backup copies of program parameters
10. Preserve parameters between tool executions

Appendix F - Stand Alone SAS program “JOB_01_Write_data_to_Excel_File.sas”

This SAS Code performs the following general tasks:

1. Includes the global macro variable parameters
2. Defines directory paths
3. Defines VBS call parameters
4. Creates an Excel output file
5. Calls the SAS X command to run the VBS script to execute Excel (after SAS runs)

Appendix G - Stand Alone VBS program "VBS_Execute_script.vbs"

This VBA Code performs the following general tasks:

1. Accept parameters from the SAS program
2. Starts execution of Excel
3. Opens an Excel workbook
4. Includes VBA common subroutines for use by the VBA formatting routine
5. Includes and executes the VBA formatting routine
6. Deletes the formatting routines (VBA MACROS) when formatting is complete
7. Saves the file as an Excel *.xls file (this can convert an *.XLM file to a *.xls file)

Appendix H - Stand Alone VBS program "Dept_Common.bas"

This VBA Code performs the following general tasks:

1. VBA Subroutine to put a bold border around selected cells
2. VBA Subroutine to freeze the top row and freeze the top row for printouts

Appendix I - Stand Alone VBS program "JOB_01.bas"

This VBA Code performs the following general tasks:

1. VBA code formats output Excel workbook from JOB_01_Write_data_to_Excel_File.sas
2. VBA Subroutine uses both subroutines from Dept_Common.bas

CONCLUSION

Over the last several years individual pieces and parts of this tool have been cropping up in my papers. Efforts to join them together have finally been successful. The space limitations of this paper only allowed me to present the tool as a high level design of the final product. This tool can provide a company with an economical solution to allow more users access to the power of SAS software. As presented here each user requires PC SAS to be installed on their computer. I do not know if setting up calls to a remote server would be feasible or less expensive, but I am sure that a BASE SAS license may be cheaper than SAS® Enterprise Guide®, SAS® Integration Technologies, or SAS® BI Server software. Of course this tool does not fully replace the functionality of any of these SAS Software 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
Address: P.O.Box 42434
City, State ZIP: Phoenix AZ, 85080
Work Phone: 623-337-0269
E-mail: William@owlcomputerconsultancy.com

Appendix A - Common Excel and Excel User Form terms

Term	Structure	Use in the Tool
Safe Mode	A method on executing Excel from a command line within a Windows *.bat file	This will start Excel using command switches that turn off some of the features of Excel.
Integrated Development Environment (IDE)	Excel screen that appears when you type the Alt+F11 keys with a spreadsheet open.	This is the Excel tool we will used to build our reporting tool to run our SAS programs
User Form (spelled UserForm)	A program “Object” used by Excel to store tool components	Used to display the components of the tool
Label	A component of a UserForm that stores text that cannot be changed.	UserForm field describing another Form component, used to describe screen fields.
TextBox	A component of a UserForm that stores text that can be changed.	A box on the UserForm that contains a value that can be changed; this value is used later. We will enter fields or show data we update.
ComboBox	A component of a UserForm that stores a list of text values that can be selected. To provide lists of valid option values.	We will use this to store a list of valid programs and prevent the user from picking an invalid value.
CommandButton	A component of a UserForm that when pressed activates an action.	The Command buttons will be used to initiate some action requested by the user (You).
CheckBox	A component of a UserForm that toggles between the on and off state.	Here we will use this to toggle the on/off state of a requested condition.
Properties Window	A window on the Integrated Development Environment (IDE) that shows the state of each element of an item on the UserForm	We will use this window to associate data elements on an Excel spreadsheet with text fields on the UserForm.
PropertySheet	A window on the Integrated Development Environment (IDE) that shows the state of each element of an item on the UserForm	Property sheets associate Excel spreadsheet data elements with text fields on the UserForm, or apply values to the item
ControlSource	A field on the PropertySheet that points to an Excel Cell that is used to store the contents of the item at runtime.	We will use this to point to where the data value for this UserForm item is stored in the Excel Spreadsheet.
Name	A field on the Property Sheet that describes the name of a component.	We will not use this field for most items on the UserForm, making the component items and types clearly identifiable. However, some items on the UserForm will have the name changed so we can reference specific spreadsheet values, cell, or other items without special coding.

Table 1. Glossary of terms used with Excel Intergrated Development Environment.

Appendix B - Map of controls in the Tool-Box pop-up window

IDE Controls on the Toolbox Pop-up Menu			
Menu option to select more controls	Label	TextBox	ComboBox
ListBox	CheckBox	OptionButton	ToggleButton
Frame	CommandButton	TabStrip	MultiPage
ScrollBar	SpinButton	Image	RefEdit

Table 2. Description of Tool-Box items, not all of these items shown here are used in the tool.

Appendix C –Excel Menu as shown in the Excel IDE.

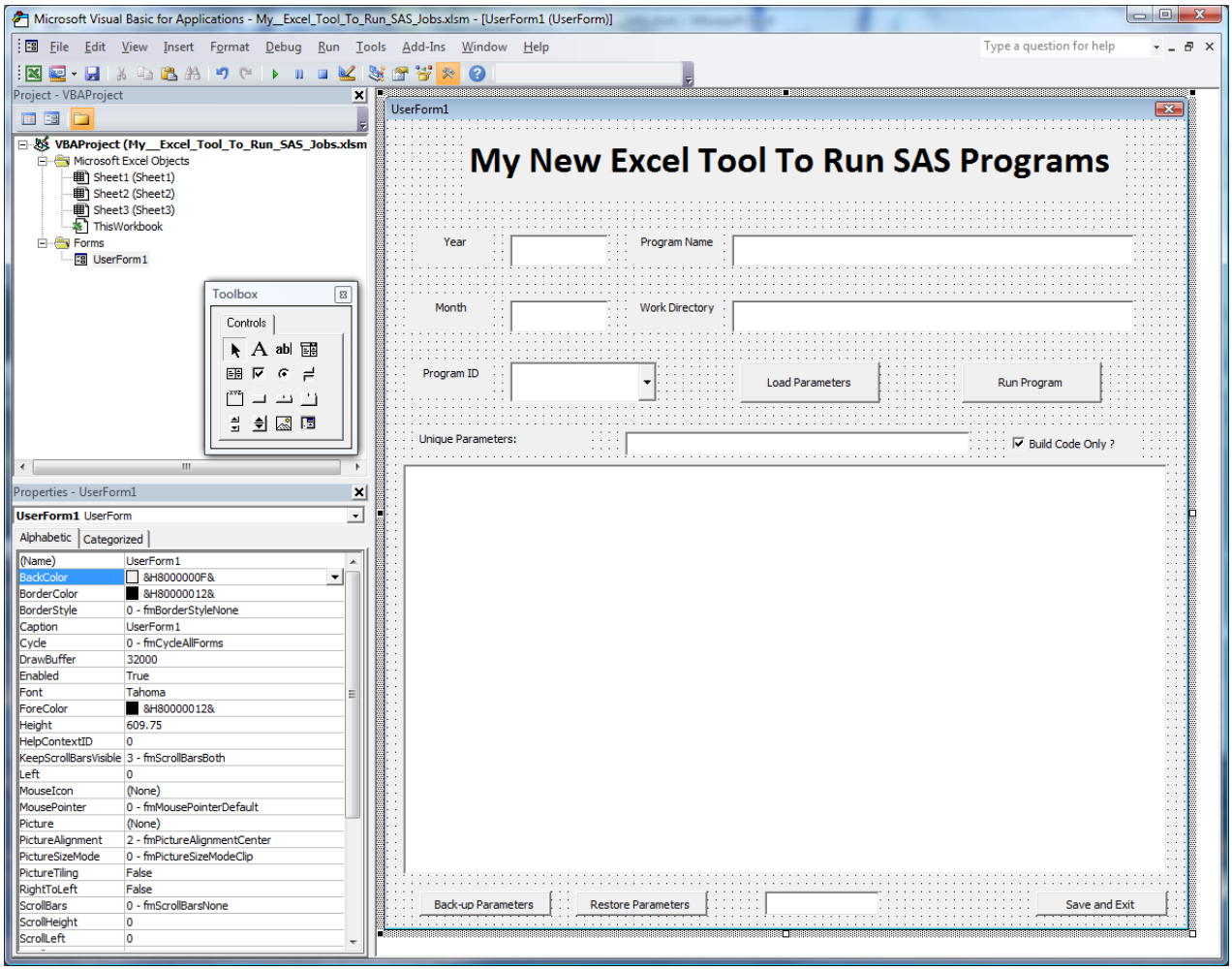

Figure 6. Tool menu shown in the Excel Intergrated Development Environment.

Appendix D - Table of descriptions of items on the Tool main menu

<table>
<tr><th colspan="4">Label1 – text -- Contains the Title, and is not used for anything else.</th></tr>
<tr><td>Label2 – text “Year”, no other use</td><td>TextBox1- Contains the value for variable Year linked to an Excel cell</td><td>Label4 – “Program Name”, no other use</td><td>TextBox3- Renamed to Program_name, Contains the value for variable Program_Name linked to an Excel cell</td></tr>
<tr><td>Label3 – text “Month”, no other use</td><td>TextBox2- Contains the value for variable Month linked to an Excel cell</td><td>Label5 – “Work Directory”, no other use</td><td>TextBox4- Contains the value for variable Work_Directory linked to an Excel cell</td></tr>
<tr><td>Label6 – text “Program ID”, no other use</td><td>ComboBox1 – Contains the list of programs installed in the tool, this is initialized by VBA code and linked to cells in the Excel workbook</td><td>CommandButton1 – “Load Parameters” this control is linked to a VBA routine that copies data from a worksheet to TextBox5 below</td><td>CommandButton2 – “Run Program” this control is linked to a VBA routine that executes the VBA code to do the processing</td></tr>
<tr><td>Label7 – text “Unique Parameters”, no other use</td><td colspan="2">TextBox7 – Renamed to Program_directory so we can take advantage of the properties of a TextBox without writing special code to do the work; we will not make this visible on the GUI screen</td><td>CheckBox1 – A flag that is true when checked to be used in the VBA code to control processing</td></tr>
<tr><td colspan="4">TextBox5- Renamed ‘Program_Parms’ to take advantage of TextBox properties without special code, this Excel control (like TextBox3, TextBox6, and TextBox7) points to cells on the JOB_xx_PARMS worksheets. The contents of the ControlSource field will be pointed to the current JOB_xx_Parms spreadsheet.</td></tr>
<tr><td>CommandButton3 – “Back-up Parameters” this control is linked to a VBA routine that copies data from TextBox5 to a backup worksheet</td><td>CommandButton4 – “Restore Parameters” this control is linked to a VBA routine that copies data from a backup worksheet to TextBox5</td><td>TextBox6 – Renamed to Program_Number so we can take advantage of the properties of a TextBox without writing special code to do the work. We will not make this visible on the GUI screen.</td><td>CommandButton5 – “Save and Exit” this control is linked to a VBA routine that saves the contents of the workbook and exits Excel. Because VBA runs a macro there are always Workbook changes</td></tr>
</table>

Table 3. Description of items used to build the tool menu (UserForm1).

Appendix E - VBA Code for Excel Workbook Tool to run SAS as an Application

```
'Part 1 - Step 1: VBA code to initialize the UserForm1 object called ComboBox1:
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  My Company Name and software Header info goes Here
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  Subroutine name    : UserForm_Initialize
''  Subroutine function:
''
''  Load the initial values into Excel User Form "UserForm1"
''  on the full screen
''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  Parameters: None
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  last update:
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim ParmsAreLoaded As Boolean, parm_file As String
Dim bat_file As String, Full_Path As String

Private Sub UserForm_Initialize()
    ' establish a base random number
    Randomize

    ' hide the TextBox6 (named Program_number)at startup
    Program_Number.Visible = False
    Program_directory.Visible = False

    ' set the Build code only flag to true at startup
    Worksheets("Control_info").Range("A6").Value = True

    ' set a flag to describe the status of the parms
    ParmsAreLoaded = False ' initialize to false

    ' count the number of job names in the list for ComboBox1
    Max = 0
    Do Until test = "###"
        idx = 4 + Max
        i = Trim(Str(idx))
        test = Worksheets("Control_info").Range("C" + i).Value 'skip 1st space
        Max = Max + 1
        If test = "" And idx > 4 Then test = "###"
    Loop

    ' Load the job names into ComboBox1
    idx = 4          ' Cell C4 - the first ComboBox1 value
    Do Until idx > Max + 2
        i = Trim(Str(idx))
        UserForm1.ComboBox1.AddItem (Worksheets("Control_info").Range("c" + i).Value)
        idx = idx + 1
    Loop
End Sub
```

```
'Part 1 - Step 2: ComboBox1 displays the Unique Parameters in the TextBox5
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  My Company Name and software Header info goes Here
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  Subroutine name    : CommandButton1_Click
''  Subroutine function:
''
''  Verify Selected Program name and Load Program
''  Unique Parameters
''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  Parameters: None
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  last update:
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Private Sub CommandButton1_Click()

    Dim Pgm_number As String
    Program_Number.Text = Worksheets("Control_info").Range("C3").Text
    Pgm_number = Program_Number.Value + "_Parms"


    Path = False
    Full_Path = ""

    Select Case Pgm_number
    Case "JOB_01_Parms"
        Call fix_parms(Pgm_number)
    Case "JOB_02_Parms"
        Call fix_parms(Pgm_number)
    Case Else
        Pgm_number = "Blanks"
        Call fix_parms(Pgm_number)
    End Select

End Sub

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  My Company Name and software Header info goes Here
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  Subroutine name    : fix_parms
''  Subroutine function:
''
''  Subroutine called by "Load_Parms" to initialize
''  data fields
''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  Parameters: Pgm_number - a string with the name of the Program
''                           that has been selected for processing
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  last update:
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Sub fix_parms(Pgm_number As String)
```

```
    Program_Parms.ControlSource = Pgm_number + "!a1"      ' Unique Parms
    Program_directory.ControlSource = Pgm_number + "!a2"  ' Working Directory
    Program_name.ControlSource = Pgm_number + "!a3"       ' SAS Program Name

    ParmsAreLoaded = True ' reset to true

    parm_file = Program_Number.Value + "_parm_Code.sas"
    bat_file = "run_" + Program_Number.Value + ".bat"

    Full_Path = TextBox4.Value + "\" + TextBox1.Value + "_" + TextBox2.Value + "\"
End Sub

'Part 1 - Step 3: This step starts with the CommandButton2_Click subroutine
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  Subroutine name    : CommandButton2_Click
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
''  Subroutine function:
''
''  Verify a program is selected and the Parameters
''  are loaded, Start SAS, Call "submit_Control_info" to
''  submit the SAS program, and exit
''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  last update:
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Private Sub CommandButton2_Click()
  If Program_name.Value = "" Then
    response = MsgBox("No Parameters loaded or Program Selected", vbCritical, _
                      "Error - Invalid Selections")
    If response = 1 Then
            GoTo my_exit
        End If
    End If

    response = MsgBox("Are You Really ready to Run the Program", vbYesNo, "Verification")
    If response = vbNo Then
        GoTo my_exit
    End If
    If response = vbYes Then
        GoTo my_program
    End If

    GoTo my_exit

my_program:

    If Worksheets("Control_info").Range("a6").Value = False Then
        Dim SAS As Object

        ' start SAS
        Set SAS = CreateObject("SAS.Application")
        SAS.Visible = True
        SAS.Wait = True
    End If
```

```
    Call submit_Control_info(SAS)


my_exit:
End Sub
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  Subroutine name    : submit_Control_info
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
''  Subroutine function:
''
''  Subroutine called by "CommandButton2_Click"
''  to build and submit the SAS code files
''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  last update:
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub submit_Control_info(SAS As Object)

    ' future use
    Dim sas_parm As String
    Dim sas_vari As String

    ' current use
    Dim My_Random_1 As String
    Dim My_Random_2 As String
    Dim new_code_file As String
    Dim sas_code As Variant
    Dim sas_code_file As String
    Dim sas_command As Variant
    Dim sas_dir As String
    Dim idx As String
    Dim sas_list As String
    Dim sas_log As String
    Dim sas_O As Object


    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' Build directory paths for this program
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

    ' output working directory
   Full_Path = TextBox4.Value + "\" + TextBox1.Value + "_" + TextBox2.Value + _
               "\" + Program_directory.Value

    ' input SAS Source code directory
    sas_dir = Trim(Worksheets("Control_info").Range("A15").Value)

    parm_file = Program_Number.Value + "_parm_Code.sas"
    bat_file = "run_" + Program_Number.Value + ".bat"


    ' $A6 = False means run the program now from Excel
    If Worksheets("Control_info").Range("a6").Value = False Then
```

```
        SAS.submit ("options source source2 notes mprint mlogic symbolgen; ")
    End If

    ' Generate two different random numbers
    My_Random_1 = Str(Int(((9999 - 1000 + 1) * Rnd) + 1000))
    My_Random_2 = Str(Int(((9999 - 1000 + 1) * Rnd) + 1000))

    altlog = "'%my_dir%\run_time_log_%my_counter1%_%my_counter2%.txt'"
    altprint = "'%my_dir%\run_time_output_%my_counter1%_%my_counter2%.txt'"

    sysin = "'" + Full_Path + "\" + parm_file + "'"


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ''  Output the parm file.

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' open a disk file and create a temp file to include into sas
    Open Full_Path + "\" + parm_file For Output Shared As #1

    Print #1, "*-----------------------------------------------------------*;"
    Print #1, "*                        set up the options                 *;"
    Print #1, "*-----------------------------------------------------------*;"
    Print #1, "                                                              "
    Print #1, "                                                              "
    Print #1, "*************************************************************;"
    Print #1, "* Note - Verify that all of the following options are valid   ;"
    Print #1, "* for the version of SAS you are using - Some of these are new;"
    Print #1, "*************************************************************;"
    Print #1, "                                                              "
    Print #1, "                                                              "
    Print #1, "options mautosource Number Pageno = 1 Linesize = 120 error=3  ;"
    Print #1, "options mprint mlogic mprintnest mlogicnest symbolgen source2 ;"
    Print #1, "options spool Notes source compress=binary                    ;"
    Print #1, "options nofmterr dkricond=warn                                ;"
    Print #1, "run;                                                          "
    Print #1, "                                                              "
    Print #1, "                                                              "
    Print #1, "                                                              "

    ' set up a proc printto command for the log output file
    sas_log = "filename saslog '" + Full_Path + "\" + Program_Number.Value + "_" + _
                Trim(My_Random_1) + "_" + Trim(My_Random_2) + "_log.txt';"
    Print #1, sas_log
    Print #1, "Proc Printto log=saslog new;"
    Print #1, "run;"


    ' set up a proc printto command for the listing output file
    sas_list = "filename saslist '" + Full_Path + "\" + Program_Number.Value + "_" + _
                            Trim(My_Random_1) + "_" + Trim(My_Random_2) + _
                            "_list.txt';"
    Print #1, sas_list
    Print #1, "Proc Printto print=saslist new;"
    Print #1, "run;"
```

```
' open the definition of the macro
i = 2
Do While i < Worksheets("Common_Parm_Text").Range("a1").Value
    idx = Trim(Str(i))
    sas_code = Worksheets("Common_Parm_Text").Range("a" + idx).Value
    Print #1, sas_code
    i = i + 1
Loop



''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  This area is commented out, it is a loop that outputs a list of
''  parms from the Control_info sheet columns A and B, Cell A1 would
''  have a count of the number of Parms and cells A2:Bx would have
''  the list of parms and values (Parm names i col A, Parm values in
''  col B) This code is left here for you to install, but you need
''  to make the mentioned changes to the control_info spreadsheet.
''  Think "Student Exercise" the "Control_info" spreadsheet needs
''  to be redesigned to use this feature
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' add the Control_info variables
'i = 2
'Do While i < Worksheets("Control_info").Range("a1").Value
'    idx = Trim(Str(i))
'    sas_vari = Worksheets("Control_info").Range("a" + idx).Value
'    sas_parm = Worksheets("Control_info").Range("b" + idx).Value
'    sas_command = "%let " + sas_vari + " = " + sas_parm + "; "
'    Print #1, sas_command
'    i = i + 1
'Loop
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  This area replaces the commented out code from above. It is a
''  simple brute force approach, but it works.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
sas_command = "%let year = " + Worksheets("Control_info").Range("a2").Text + "; "
Print #1, sas_command

sas_command = "%let month = " + Worksheets("Control_info").Range("a3").Text + "; "
Print #1, sas_command

' insert the unique parms for this program
sas_code = Worksheets(Program_Number.Value + "_Parms").Range("a1").Value
Print #1, sas_code

' close the macro
i = 2
Do While i < Worksheets("Common_Parm_Text").Range("b1").Value
    idx = Trim(Str(i))
    sas_code = Worksheets("Common_Parm_Text").Range("b" + idx).Value
    Print #1, sas_code
    i = i + 1
```

```
    Loop

    ' define the source code input lib
    sas_code = "Filename in_cde '" + Full_Path + "';"
    Print #1, sas_code

    ' include the source code for this program
    'sas_code = "%include in_cde(" + Worksheets(Program_Number.Value + _
                              "_Parms").Range("a1").Value + ");"
    sas_code = "%include in_cde(" + _
            Worksheets(Program_Number.Text + "_Parms").Range("a3").Value + ");"
    Print #1, sas_code

    ' Close before reopening in another mode.
    Close #1

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ''  Output the batch execution file.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    Open Full_Path + "\" + bat_file For Output Shared As #2

    Print #2, "rem ************************************************************"
    Print #2, "rem                     set up the options                      *"
    Print #2, "rem ************************************************************"
    Print #2, "set/a my_counter1=%random%"
    Print #2, "set/a my_counter2=%random%"
    Print #2, "set   my_dir=" + Full_Path

    ' Install your path to the SAS.EXE file here.
    ' The sasinit directory is not defined as a separate dir in this tool
    ' but it could be defined that way, that directory is used with
    ' the -SASINITIALFOLDER switch to allow the two or more jobs to run
    'concurrently to use remove "Full_Path ' +"
    ' from the print command listed below
    'Print #2, Chr(34) + "C:\Program Files\SAS\SASFoundation\9.2\SAS.exe" + Chr(34) + _
              " -altlog " + altlog + " -altprint " + altprint + " -sysin " + _
              sysin + " -SASINITIALFOLDER " + Full_Path ' + sasinit

    ' Install your path to the SAS.EXE file here.
    ' The sasinit directory is not defined as a separate dir in this tool
    ' but it could be defined that way, that directory is used with
    ' the -SASINITIALFOLDER switch to allow the two or more jobs to run
    'concurrently to use remove "Full_Path ' +"
    ' from the print command listed below
    Print #2, Chr(34) + "F:\Program Files\SASHome\SASFoundation\9.3\SAS.exe" + _
            Chr(34) + " -altlog " + altlog + " -altprint " + altprint _
           + " -sysin " + sysin; " -SASINITIALFOLDER " + Full_Path '+ sasinit
    Close #2

    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    '' Copy the source SAS Code from the source directory to the work area
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

    'object.Copy destination[, overwrite]
    Set sas_O = CreateObject("Scripting.FileSystemObject")
```

```
    Prod_path = Worksheets("Control_info").Range("a15").Value

    sas_code_file = Prod_path + "\" + Worksheets(Program_Number.Value + _
                  "_Parms").Range("a3").Value

    new_code_file = Full_Path + "\" + Worksheets(Program_Number.Value + _
                  "_Parms").Range("a3").Value

    overwrite = True
    sas_O.copyfile sas_code_file, new_code_file, overwrite
    Set sas_O = Nothing

    'now wait for the o/s to catch up
    My_Hour = Hour(Now())
    My_Minute = Minute(Now())
    My_Second = Second(Now()) + 5
    waitTime = TimeSerial(My_Hour, My_Minute, My_Second)
    Application.Wait waitTime
    If Worksheets("Control_info").Range("a6").Value = False Then ' run the code
        ' SAS Commands to really include the temp file and run the program
        SAS.submit ("Filename in_src '" + Full_Path + "';")
        SAS.submit ("%include in_src(" + parm_file + ");")
        SAS.submit ("run;")
        SAS.submit ("data _null_; *something at the end to force log output;")
        SAS.submit ("a=1;")
        SAS.submit ("run;")
        SAS.submit ("data _null_;")
        SAS.submit ("completed = setrc('Complete' , 1.0);")
        SAS.submit ("put completed ;")
        SAS.submit ("run;")
        SAS.submit ("proc printto print=print;")
        SAS.submit ("data _null_; *something at the end to force log output;")
        SAS.submit ("a=1;")
        SAS.submit ("run;")

        Dim seconds As Long
        seconds = 3

        ' test to see if SAS is busy, when it completes then exit SAS
        Y = 0
        Do While Y = 0
            If (SAS.busy) Then
                Call Wait_for_SAS(seconds)
            Else
                If (SAS.RC <> 0) Then
                    SAS.Quit
                    Set SAS = Nothing
                Y = 1
                End If
            End If
        Loop
    End If

No_Dir_to_use:
```

```
End Sub
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  Subroutine name    : Wait_for_SAS
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
''  Subroutine function:
''
''  Subroutine called by "Run_SAS_Program_Click" to
''  pause processing to ensure correct timing.
''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  last update:
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Private Sub Wait_for_SAS(wait_seconds As Long)
    My_Hour = Hour(Now())
    My_Minute = Minute(Now())
    My_Second = Second(Now()) + wait_seconds
    waitTime = TimeSerial(My_Hour, My_Minute, My_Second)
    Application.Wait waitTime
End Sub
'Part 1 - Step 4: Tool has built in safety features to help avoid simple errors
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  Subroutine name    : CommandButton3_Click
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
''  Subroutine function:
''
''  Stores the current "Unique Parameters",
''  SAS Program Name, and Output Program Directory
''  information in a backup worksheet
''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  last update:
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Private Sub CommandButton3_Click()

Dim Y As String, Z As String, My_A1 As String, My_A2 As String, My_A3 As String

    Y = Program_Number.Value + "_Parms"
    Z = Program_Number.Value + "_Backup"

    If Y = "_Parms" Or Z = "_Backup" _
     Then
       response = _
       MsgBox( _
       "No Parms are loaded or a data field is blank, data was not changed ", _
       vbOKOnly, _
       "Verification")
     Else
         My_A1 = Trim(Worksheets(Y).Range("a1").Value)
         My_A2 = Trim(Worksheets(Y).Range("a2").Value)
         My_A3 = Trim(Worksheets(Y).Range("a3").Value)
    End If
```

```
    If ((ParmsAreLoaded = True) And Not (My_A1 = "" Or My_A2 = "" Or My_A3 = "")) _
        Then
         If (Y <> "Blanks" And Y <> "") Then
             Worksheets(Z).Range("a1").Value = Worksheets(Y).Range("a1").Value
             Worksheets(Z).Range("a2").Value = Worksheets(Y).Range("a2").Value
             Worksheets(Z).Range("a3").Value = Worksheets(Y).Range("a3").Value
             Worksheets(Z).Range("b1").Value = "Program unique parms"
             Worksheets(Z).Range("b2").Value = "Data From UNIX dir"
             Worksheets(Z).Range("b3").Value = "SAS Program name"
        End If
        Else
            response = _
            MsgBox( _
            "No Parms loaded or data field is blank, data was not changed ", _
            vbOKOnly, _
            "Verification")
    End If

End Sub


'Part 1 - Step 5: A second safety feature to help avoid simple errors in the
Restore Parameters button
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  Subroutine name    : CommandButton4_Click
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
''  Subroutine function:
''
''  Restores from a backup worksheet the current
''  "Unique Parameters", SAS Program Name, and
''  Output Program Directory information
''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  last update:
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Private Sub CommandButton4_Click()

    Dim Y As String, Z As String, My_A1 As String, My_A2 As String, My_A3 As
String

    Y = Program_Number.Value + "_Parms"
    Z = Program_Number.Value + "_Backup"

    If Y = "_Parms" Or Z = "_Backup" _
        Then
            response = _
               MsgBox( _
               "No Parms are loaded or a data field is blank, data was not
changed ", _
               vbOKOnly, _
               "Verification")
        Else
            My_A1 = Trim(Worksheets(Z).Range("a1").Value)
```

```
            My_A2 = Trim(Worksheets(Z).Range("a2").Value)
            My_A3 = Trim(Worksheets(Z).Range("a3").Value)
    End If


If ((ParmsAreLoaded = True) And Not (My_A1 = "" Or My_A2 = "" Or My_A3 = "")) _
      Then
         If (Z <> "Blanks" And Z <> "") Then
              Worksheets(Y).Range("a1").Value = Worksheets(Z).Range("a1").Value
              Worksheets(Y).Range("a2").Value = Worksheets(Z).Range("a2").Value
              Worksheets(Y).Range("a3").Value = Worksheets(Z).Range("a3").Value
              Worksheets(Y).Range("b1").Value = Worksheets(Z).Range("b1").Value
              Worksheets(Y).Range("b2").Value = Worksheets(Z).Range("b2").Value
              Worksheets(Y).Range("b3").Value = Worksheets(Z).Range("b3").Value
          End If
        Else
          response = _
             MsgBox( _
             "No Parms loaded or data field is blank, data was not changed ", _
             vbOKOnly, _
             "Verification")
    End If

End Sub


'Part 1 - Step 6: Implements the final button the Save and Exit
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  Subroutine name    : CommandButton5_Click
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
''  Subroutine function:
''
''  Subroutine called by "CommandButton5_Click" to
''  save the tool, code, and condition then exit Excel
''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  last update:
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Private Sub CommandButton5_Click()

    ActiveWorkbook.Save
    Application.Quit

End Sub
```

Appendix F - Stand Alone SAS program “JOB_01_Write_data_to_Excel_File.sas”

```
**============================================================================**;
**           JOB_01_Write_data_to_Excel_File                                  **;
**============================================================================**;
%let Run_Time_Audit_Value =  JOB_01_Write_data_to_Excel_File ;
%Init_Global_Macro_Variables;


* use file directory name pieces from the Parameters set up above;
%let my_VBA   = Q:\My_BAS_Files;
%let my_path  = Q:\My_Monthly_Projects\&year._&month.;
%let my_excel = &my_path.\JOB_01_Write_data_to_Excel_File\&Excel_File_name.;

%let vbs_code      = &my_VBA.\Execute_script.vbs;* VBA subroutine to execute;
%let Input_Excel   = &my_excel;                  * File path and Input file ;
%let output_excel  = &my_excel;                  * File path and Output file;
%let bas_code_path = &my_VBA.\;                  * File path Location of bas;
%let vba_module    = JOB_01;                     * VBA Path/Module (no bas) ;
%let vba_code      = Format_JOB_01;              * VBA subroutine to execute;

**============================================================================**;
**           To execute on a 64-bit computer use the following                **;
**============================================================================**;
PROC EXPORT DATA=SASHELP.SHOES
outfile="&my_excel."
DBMS=EXCELCS
REPLACE;
Sheet=&Sheet_name;
run;

**============================================================================**;
**           To execute on a 32-bit computer use the following                **;
**============================================================================**;
*PROC EXPORT DATA=SASHELP.SHOES
*outfile="&my_excel."
*DBMS=EXCEL
*REPLACE;
*Sheet=My_Page;
*run;

option NOQUOTELENMAX; * turn off error message about length of a quoted string;

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

option QUOTELENMAX;   * turn on error message about length of a quoted string;

run;
```

Appendix G - Stand Alone VBS program “VBS_Execute_script.vbs”

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

objxl.DisplayAlerts = wdAlertsNone                         'Turn off error messages

if vba_module <> "" then
   vbCom.Import ("" & bas_code_path & vba_module & ".bas") 'Import VBA Report module
   vbCom.Import ("" & bas_code_path & "Dept_Common.bas")   'Import Common Routines

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

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

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

end if

'Save as Excel workbook
if Input_Excel <> output_excel then objxl.ActiveWorkbook.SaveAs output_excel, 56 'xlExcel8
if Input_Excel = output_excel then objxl.ActiveWorkbook.Save

objxl.Workbooks.Open(output_excel).Close
objxl.Quit
set objxl = nothing
set objwk = nothing
```

Appendix H - Stand Alone VBA program “Dept_Commom.bas”

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

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

'=========================================================
' This subroutine freezes the top row and sets the top
' row to be output on every print page.
'=========================================================
' Input Parameters = None
'=========================================================
'Update History:
'=========================================================

Sub freeze_top_row_and_print_header()

    '''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' set top row top freeze at the top of each page
    '''''''''''''''''''''''''''''''''''''''''''''''''''''''
    With ActiveWindow
        .SplitColumn = 0
        .SplitRow = 1
    End With

    ActiveWindow.FreezePanes = True

    '''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' set top row top print at the top of each page
    '''''''''''''''''''''''''''''''''''''''''''''''''''''''
    With ActiveSheet.PageSetup
        .PrintTitleRows = "$1:$1"
        .PrintTitleColumns = ""
    End With
End Sub
```

Appendix I - Stand Alone VBA program for “JOB_01.bas”

```
Attribute VB_Name = "JOB_01"
''=======================================================
''  My Company Name and software Header info goes Here
''=======================================================
''  Subroutine name    : Format_JOB_01
''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  Subroutine function: format the report for JOB_01
''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''  last update:
''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub Format_JOB_01()
    Dim myRange As Range
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' Code to define the corners of all used cells on the worksheet
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' find number of rows & columns the selcetion
    '           (a,b first row/col: x,y last row/col)
    ' Top left  cornsr = cells(a_TL_Row, b_TL_Col)
    ' bot right corner = cells(a_TL_Row + x_Row_ct, b_TL_Col + y_Col_ct)
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ActiveSheet.UsedRange.Select
    Set myRange = Selection
    a_TL_Row = myRange.Row
    b_TL_Col = myRange.Column
    x_Row_ct = myRange.Rows(myRange.Rows.Count).Row
    y_Col_ct = myRange.Columns(myRange.Columns.Count).Column
    ' use the cell selection above for these routines
    Call make_thick_outside_border
    Call freeze_top_row_and_print_header
    Columns("A:A").ColumnWidth = 25
    Columns("B:B").ColumnWidth = 17
    Columns("C:C").ColumnWidth = 15
    Columns("D:G").ColumnWidth = 12
    Today = Month(Date) & "-" & Day(Date) & "-" & Year(Date)
    With ActiveSheet.PageSetup
        .Orientation = xlPortrait
        .LeftMargin = Application.InchesToPoints(0.25)
        .RightMargin = Application.InchesToPoints(0.25)
        .TopMargin = Application.InchesToPoints(0.5)
        .BottomMargin = Application.InchesToPoints(0.5)
        .HeaderMargin = Application.InchesToPoints(0.25)
        .FooterMargin = Application.InchesToPoints(0.25)
        .LeftFooter = "&""Calibri""&9 " & Today & ""
        .CenterFooter = "&""Calibri""&9&P"
        .RightFooter = "&""Calibri""&9Prepared by my_Department"
        .LeftHeader = ""
        .CenterHeader = "&""Calibri,Bold""&14My New Report"
        .RightHeader = ""
        .Zoom = 85
        .CenterHorizontally = True
    End With
    Range("A1:A1").select
End Sub
```