Paper 8680-2016

Integrating Microsoft® VBScript and SAS®

Christopher Johnson, BrickStreet Insurance

ABSTRACT

VBScript and SAS are each powerful tools in their own right. These two technologies can be combined so that SAS code can call a VBScript program or vice versa. This gives a programmer the ability to automate SAS tasks, traverse the file system, send emails programmatically, manipulate Microsoft® Word, Excel, and PowerPoint files, get web data, and more. This paper will present example code to demonstrate each of these capabilities.

Contents

INTRODUCTION

The SAS software most notably uses the SAS language; however, it also allows the use of SAS Macro language, SQL, SCL, C, as well as utilizing VBScript. If you have ever tried to use the Automation functionality in SAS Enterprise Guide, in the background, it creates a VBScript designed to kick off a SAS program and places the script in the Windows task scheduler. We can use this as a springboard to extend the power of both software platforms.

GETTING STARTED

VBScript is probably the easiest language to get begin writing, and it is free. It runs natively in Windows, and it is interpreted by the Windows Scripting Host (Microsoft, 2014), so there is nothing to install. To get started, you need to:

1. Create a text file.
2. Change the extension of the file from .txt to .vbs.
3. Enter your code in the file.
4. Save and close the file.
5. Double click the resulting file to run the script.

Once you have a script file to work in, you can define variables, create functions, and use normal programming concepts such as loops, conditional logic, etc. More importantly, you can define objects that represent an instance of Word, Excel, PowerPoint, Access, Outlook, SAS, HTML, the file system, and much more. These objects have methods, properties, and functions designed to manipulate the object.

To make the code easier to read, I will use certain conventions. Variables will be lowercase and shown in red. Objects will be lowercase and shown in blue and starting with the letter o. Functions will be sentence case. Each sample program will begin with the program name, commented out with an apostrophe so that it is ignored when the program runs. SAS keywords will be shown in all caps and color-coded according to SAS Enterprise Guide standards. Lastly, we will refer to VBScript code as the script and SAS code as the program.

VBScript allows the user to choose whether you will declare your variables before using them or not. While it is not required, best practices are to start your scripts with the following line:

```
Option Explicit
```

This tells the script that you will declare all variables before they are used. You then list each variable preceded by the keyword Dim. We will exclude this in all code presented to make the code as concise and easy to read as possible.

For additional language help, W3Schools offers an excellent reference (W3Schools, n.d.). Another useful resource are the script examples from ActiveXperts (ActiveXperts, n.d.).

VBSCRIPT RUNNING SAS

There are many reasons why a user might want to call a SAS program from a VBScript. Once I have a well-developed SAS program, I will often hand it off to other users to run periodically. Those users may not be proficient in SAS or may not know how to run SAS at all. I can write a VBScript that takes input from the user, calls a SAS program, passes the parameters, and saves the results. The user needs no technical knowledge in order to run the program. In the script, I can also create a log file that documents when and who runs the program.

Another use of the VBScript is to clean the data before SAS begins processing. In cases where in INPUT statement is already complicated, if the programmer wants to apply a simple rule such as replacing a particular character or string, this may be easier to handle before passing the file to SAS. Similarly, there may be situations that would cause SAS to not be able to read a file such as nonstandard characters. These can also be handled prior to running the file.

CREATING AND RUNNING CODE

The following VBScript demonstrates how to create and run a SAS program from scratch. It first defines a string that will hold our SAS code. It then creates a SAS Application object called oapp. It uses the New method of that object to create a SAS Project object called oprj. It uses the Add method to create a Code object called ocode. It uses Text property to set the code to our str variable. The Run, Close, and Quit methods are called to execute the code and close the object:

```
'RunSASCode.vbs

str = "%DoSomething(""\\Server\InFile.txt"", ""\\Server\OutFile.txt"");"

RunCode(str)

Sub RunCode(codestr)
    Set oapp = CreateObject("SASEGObjectModel.Application.4")
    Set oprj = oapp.New()
    Set ocode = oprj.CodeCollection.Add
    ocode.Text = codestr
    ocode.Run
    oprj.Close
    oapp.Quit
End Sub
```

RUNNING EXISTING CODE

The following code is an extension of the methods explained by Chris Hemedinger (Hemedinger, 2012). This example demonstrates a more real-world scenario using an existing SAS Enterprise Guide project. The script prompts the user for a value. It then creates and opens an instance of SAS, and loops through all parameters. If one is found with the right name, it sets it to the value we obtained from the user. It then searches through the code collection for the desired code files, running each in turn. After running a program, it saves the log, code, datasets and results to a specified folder. It completes by saving and closing the program.

This file can serve as a template for automating any SAS Enterprise Guide project. The user only needs to customize the parameter calls, project name, and code files needed. All resulting logs and results are available for the user to review outside SAS or to work with programmatically:

```
'RunSASProgram.vbs

'Set ErrorHandler

On Error Resume Next

server = "\\server\"
projname = server & "Testing.egp"
codefiles = Array("LIBNAMES", "WORK")

val = InputBox("Enter Value", , "1")
If val = "" Then WScript.Quit

Set oapp = CreateObject("SASEGObjectModel.Application.4.3")
Set oproj = oapp.Open(projname,"")
Set oparameters = oproj.Parameters
For Each oparameter In oparameters
    If oparameter.Name = "VALUE" Then oparameter.Value = val
Next
Set ocodecollect = oproj.CodeCollection
For Each code In codefiles
    For Each ocode In ocodecollect
          If ocode.Name = code Then
                ocode.UseApplicationOptions = False
                ocode.GenListing = True
                ocode.GenSasReport = False
                ocode.Log.SaveAs server & "Results\" & ocode.Name & ".log"
                ocode.Run
                ocode.SaveAs server & "Results\" & ocode.Name & ".sas"
                For n=0 to (ocode.OutputDatasets.Count -1)
```

```
                    dataName = ocode.OutputDatasets.Item(n).Name
                    ocode.OutputDatasets.Item(n).SaveAs server &
"Results\" & dataName & ".xls"
                Next
                For n=0 to (ocode.Results.Count -1)
                    ocode.Results.Item(n).SaveAs server & "Results\" &
WScript.ScriptName & n & ".lst"
                Next
            End If
        Next
Next
oproj.Save
oproj.Close
oapp.Quit

'Reset Error Handler
On Error Goto 0
```

For a comprehensive guide to the SAS Object Model, see the SASEGScripting Help file (Support.SAS.com).

RUNNING SAS IN BATCH MODE

Using a server based SAS Enterprise Guide, a user losses the ability to submit a SAS batch job. This can be overcome using VBScript and has been demonstrated by Chris Hemedinger (Hemedinger, 2012):

```
'RunSASBatch.vbs

'Create an object and instance of SAS.
Set oapp = CreateObject("SASEGObjectModel.Application.4.3")
oapp.SetActiveProfile("Chris")
Set oproj = oapp.New
Set osasprog = oproj.CodeCollection.Add
osasprog.UseApplicationOptions = False
osasprog.GenListing = True
osasprog.GenSasReport = False
osasprog.Server = "SASApp"

'Set the code to run.
osasprog.Text = "DATA testme; SET SASHELP.CLASS; RUN;"
osasprog.Text = osasprog.Text & " PROC MEANS DATA=testme; RUN;"
osasprog.Run

'Save the log file
osasprog.Log.SaveAs "\\server\" & WScript.ScriptName & ".log"

'Save the output and results
For n=0 to (osasprog.OutputDatasets.Count -1)
    dataName = osasprog.OutputDatasets.Item(n).Name
    osasprog.OutputDatasets.Item(n).SaveAs "\\server\" & dataName & ".xls"
Next
For n=0 to (osasprog.Results.Count -1)
    osasprog.Results.Item(n).SaveAs "\\server\" & WScript.ScriptName &
".lst"
Next

Application.Quit

'Run at the command line:
```

```
'cscript RunSASBatch.vbs
```

This script is called from the command prompt, which you can access in Windows by clicking Start and typing cmd in the search box.

SAS RUNNING VBSCRIPT

Traditional SAS users may feel more at home in creating SAS programs that call VBScripts. The following program demonstrates how to create the script from within SAS. The FILENAME statement sets the script name and location. Then, a DATA step creates the file, and writes out a line of code for every PUT statement. Finally, the X command runs the program.

One note of caution. Any SAS instance should be able to create the script. However, in some situations such as a server running SAS Enterprise Guide, the SAS Administrator may have disabled the X command. In that case, your options are to have that enabled or to create the script in SAS and run it manually.

In later sections, we will examine nontrivial uses of this technique:

```
/*Create VBScript File*/

  FILENAME script "\\Server\FileName.vbs";

  DATA __NULL__;
      FILE script;
      PUT 'msgbox "Hello World!"';
  RUN;

  X script;
```

CREATING AND MANIPULATING WORD AND EXCEL FILES

Microsoft Word, Excel, PowerPoint, and Access are among the most frequently used applications in the world. In this paper I will demonstrate creating and updating Word, Excel, and PowerPoint programmatically. Since SAS has powerful methods of interacting with Access databases, we will omit it from this discussion.

To learn more about the syntax of code specific to one of the Microsoft Office products, it is often useful to record a macro, perform the tasks you would like to replicate, and then stop the macro recording. You can then view the macro and see exactly how the program accomplished your task. Macros technically use VBA or Visual Basic Applications, which is slightly different that VBScript, but generally, you can easily translate between the two.

The next few sections will contain strictly VBScript code, which can be created and run in a SAS session.

WORD

Text reports are most often stored as Word documents. Situations may arise where a frequently used report needs to be created or updated periodically, with the structure remaining constant. The following script demonstrates the use of this technique using a trivial example. A string is stored in the variable msg. A file system object is created to get the current directory of the script. A Word object is created, styles are set, and the text is entered. The file is then saved to the current directory and closed:

```
'Word.vbs

msg = "This is a test."

'Get Current Directory
Set ofilesys = CreateObject("Scripting.FileSystemObject")
Set oscript = ofilesys.GetFile(WScript.ScriptName)
Set ofolder = ofilesys.GetFolder(oscript.ParentFolder)
currentpath = ofolder.path + "\"

'Create Word File
Set oword = CreateObject("Word.Application")
```

```
oword.Caption = "Test Caption"
oword.Visible = False
Set odoc = oword.Documents.Add()
Set oselection = oword.Selection

'Type Message
oselection.Font.Name = "Arial"
oselection.Font.Size = "12"
oselection.Font.Bold = False
oselection.TypeText msg
oselection.TypeParagraph()

'Save and Close Word File
odoc.SaveAs(currentpath + "test.doc")
oword.Quit
```

Additionally, the programmer may insert images in the document.

WORD WITH BOOKMARKS

You may not want to create a Word document from scratch. To edit an existing document, you can add bookmarks at the places that you want to edit, and then programmatically change them:

```
'WordBookmarks.vbs

'Get Current Directory
Set ofilesys = CreateObject("Scripting.FileSystemObject")
Set oscript = ofilesys.GetFile(WScript.ScriptName)
Set ofolder = ofilesys.GetFolder(oscript.ParentFolder)
currentpath = ofolder.path + "\"

'Create Word File
Set oword = CreateObject("Word.Application")
oword.Visible = False
Set odoc = oword.Documents.Open(currentpath + "Test.docx")
Set orange = odoc.Bookmarks("Change").Range
orange.text = "test"
odoc.Bookmarks.Add "Change", orange

'Save and Close Word File
odoc.SaveAs(currentpath + "Test.docx")
oword.Quit
```

One point to notice is that after you edit a bookmark's text, the bookmark is deleted. In order to keep the bookmark, since we already have the range saved in the orange object, we only need to recreate the bookmark with the same name and the saved range.

EXCEL

The following script creates an Excel file, makes the instance of Excel invisible to the user, and then enters data, saves, and closes:

```
'Excel.vbs

'Get Current Folder
Set ofilesys = CreateObject("Scripting.FileSystemObject")
Set oscript = ofilesys.GetFile(WScript.ScriptName)
Set ofolder = ofilesys.GetFolder(oscript.ParentFolder)
currentpath = ofolder.path + "\"
```

```
'Create Excel File
Set oexcel = CreateObject("Excel.Application")
oexcel.Visible = False
Set oworkbook = oexcel.Workbooks.Add()

'Add Formatting and Text
oexcel.Cells(1, 1).Value = "Text"
oexcel.Cells(1, 1).Interior.Color = 65535

'Excel Constants
xlNone = -4142
xlContinuous = 1
xlThick = 4
xlThin = 2

'Select Range and Add Borders
Set orange = oexcel.Range("B3:C5")
orange.Borders.LineStyle = xlContinuous
orange.Borders.Weight = xlThin

'Loop Through Cells
For i = 0 To 2
    For j = 0 To 1
          oexcel.Cells(3,2).Offset(i,j) = i*j
    Next
Next

'Save and Close Excel File
oworkbook.SaveAs(currentpath & "Test.xlsx")
oexcel.Quit
```

There are a few points here to note. In Excel, once we have the oexcel object, we can select and set the value or formatting of one of its Cells or a Range. You can also work with an Offset of a Cell, which is useful if you need to use a For or While loop.

Many tools already exist in SAS for dealing with Excel files, particularly when the file represents a simple table. However, when the Excel file contains a complex report, whose structure is not strictly tabular, this scripting technique can be used to update it. The syntax for working with Excel follows VBA or Visual Basic Applications. Again, more examples can be found online or by using Excel's macro functionality.

POWERPOINT

The following script demonstrates how to programmatically create a PowerPoint object, add a slide, add text, save the file to the current directory, and close:

```
'PowerPoint.vbs

'Get Current Directory
Set ofilesys = CreateObject("Scripting.FileSystemObject")
Set oscript = ofilesys.GetFile(WScript.ScriptName)
Set ofolder = ofilesys.GetFolder(oscript.ParentFolder)
currentpath = ofolder.path + "\"


Set oppt = CreateObject("PowerPoint.Application")
oppt.Visible = True
Set opres = oppt.Presentations.Add
Set oslide = opres.Slides.Add(1, 2)
```

```
oslide.Shapes(1).TextFrame.TextRange.Text = "My first slide"
oslide.Shapes(2).TextFrame.TextRange.Text = "This is some text."

opres.SaveAs(currentpath + "testppt.ppt")
oppt.Quit
```

SENDING EMAILS FROM SAS

If a program runs periodically, a user might benefit from an email stating that it completed, maybe even containing the results. A programmer might like to get an email alert if a program generates an error. The following sections show how to create such an email using either Microsoft Outlook or a web email service.

OUTLOOK

The following script creates and sends an email using Outlook. The programmer can set the To, Subject, Body, and even an Attachment. The subroutine Email then creates and sends the email:

```
'EmailOutlook.vbs

'Set Variables
mailto = "me@gmail.com"
subject = "Test Email"
body = "This is an VBScript - Outlook Email Test."
attachment = "k:\test.jpg"
Email mailto, subject, body, attachment

'Function to Generate Email
Sub Email (mailto, subject, body, attachment)
    Set oapp = CreateObject("Outlook.Application")
    Set oitem = oapp.CreateItem(0)
    With oitem
          .To = mailto
          .Subject = subject
          .HTMLBody = body
    End With
    Set omsgattachments = oitem.Attachments
    omsgattachments.Add attachment
    oitem.Send
End Sub
```

SMTP

Similar to the Outlook scenario, the following script creates an email using an SMTP email service. Many online email services such as Gmail and Yahoo can be automated in this way. The user may need to log into the service and enable POP for this to work. One drawback is that the programmer will need to hardcode or prompt the user for the username and password:

```
'EmailSMTP.vbs

'Set Variables
mailto = "me@gmail.com"
from = "me@gmail.com"
subject = "Test Email"
body = "This is an VBScript - Outlook Email Test."
username = "myusername"
password = "mypassword"
Email mailto, from, subject, body, username, password

'Function to Generate Emails
```

```
Sub Email (mailto, from, subject, body, username, password)
    Set omessage = CreateObject("CDO.Message")
    With omessage
          .Subject = subject
          .From = from
          .To = mailto
          .TextBody = body
    End With
    With omessage.Configuration.Fields
     .Item("http://schemas.microsoft.com/cdo/configuration/sendusing") = 2
     .Item("http://schemas.microsoft.com/cdo/configuration/smtpserver") =
"smtp.gmail.com"

    .Item("http://schemas.microsoft.com/cdo/configuration/smtpauthenticate"
) = 1
    .Item("http://schemas.microsoft.com/cdo/configuration/sendusername") =
username
    .Item("http://schemas.microsoft.com/cdo/configuration/sendpassword") =
password
    .Item("http://schemas.microsoft.com/cdo/configuration/smtpserverport")
= 25
     .Item("http://schemas.microsoft.com/cdo/configuration/smtpusessl") =
True

    .Item("http://schemas.microsoft.com/cdo/configuration/smtpconnectiontim
eout") = 60
     .Update
    End With
    omessage.Send
End Sub
```

SEARCHING FOR SAS CODE

This is one of my most common uses for VBScript in working with SAS. Unlike many of the proceeding examples, this code is not a template. It is meant to save in a VBScript file and run as is.

I keep all of my .sas files in a single folder. Even when using SAS Enterprise Guide, I save the code files to my Code folder. This gives me several benefits such as having the ability to backup and restore those files independently. Since a .sas file is really plain text, it also permits searching.

The following script asks the user to select a folder. It then prompts the user for a string. The script attempts to open every file in the folder. Once opened, it loops through every row. If the string is found in that row, the script exports the row (along with the name of the file and row number) to a file called results.txt. When finished, it alerts the user and displays the number of times it found the string.

How is this useful? If I have created a SAS program, maybe years ago, and I need to find that code to run it again, I only need to remember an uncommon word or phrase that I used to find it again. Similarly, if I want to find an example where I used a SAS technique to get the syntax, I only need to use this tool to find the file and row number:

```
'SearchDetail.vbs

'Set Error Handler
On Error Resume Next

'Define Objects
Set oshell = CreateObject("Shell.Application")
Set ofilesys = CreateObject("Scripting.FileSystemObject")
Set oscript = ofilesys.GetFile(WScript.ScriptFullName)
Set oargs = WScript.Arguments
```

```
'Get Folder
If oargs.Count >= 1 Then
    opath = oargs (0)
Else
    Set ofolder = oshell.BrowseForFolder(0, "Select a folder:", 0,
oscript.parentfolder&"\")
    Set ofolderitem = ofolder.Self
    opath = ofolderitem.Path
    If Err <> 0 Then WScript.Quit
End If

'Get Search String
If oargs.Count >= 2 Then
    searchstr = oargs(1)
Else
    searchstr = Inputbox("Enter a search string:")
End If
If searchstr = "" then WScript.Quit

'Search Text Files For String
Set ofolder = ofilesys.GetFolder(opath)
Set ofiles = ofolder.files
count = 0
Set oresults = ofilesys.CreateTextFile
(oscript.parentfolder&"\results.txt", True)
For Each ofile In ofiles
    line = 0
    Set ofileinst = ofilesys.OpenTextFile(ofile, 1)
    Do Until ofileinst.AtEndOfStream
                str = ofileinst.Readline
          line = line + 1
          If (InStr(LCase(str), LCase(searchstr)) > 0) Then
                count = count + 1
                oresults.WriteLine("File: " & ofile.path & ", Line: " &
line)
                oresults.WriteLine(str)
          End If
    Loop
    File.Close
Next
If count = 0 Then msgbox("String not found.") Else msgbox(count & "
occurrences found.")

'Close File
oresults.Close

'Reset Error Handler
On Error Goto 0
```

Another use for this program is to search data files. I work with daily text-formatted data files. I store these files in a single folder, so that they are searchable with this script. It is easy to accumulate hundreds or thousands of file in this way, and they would not be easily searched otherwise.

GETTING WEB DATA

If you want to automate data extraction from a web source, you can create an HTTP object to get the data. You can use an API for some webpages or customize the web address to pass your parameters to the website. For instance, you can get Googles stock info by getting the webpage and passing the format (i.e. csv) and stock (i.e. GOOG) in the

address. The HTTP object then returns the text, which can be passed to SAS for processing. The following script gets the Google stock quotes, Yahoo weather, and a sample Google search for demonstration:

```
'APIs.vbs

'Create Objects
Set ohttp = CreateObject("MSXML2.XMLHTTP")
Set ohttp2 = CreateObject("Msxml2.ServerXMLHTTP")

'Get Data and Display
'Google Finance
url="http://www.google.com/finance/historical?output=csv&q=GOOG"
Call ohttp.Open("GET", url, FALSE)
ohttp.Send
msgbox ohttp.ResponseText

'Yahoo Weather
url="http://xml.weather.yahoo.com/forecastrss?p=25301"
Call ohttp.Open("GET", url, FALSE)
ohttp.Send
WScript.Echo(ohttp.ResponseText)

'Google Search
url ="http://www.google.com/search?q=hello+world"
ohttp2.Open "GET", url, False
ohttp2.Send
WScript.Echo ohttp2.ResponseText

'Close Objects
Set ohttp = Nothing
Set ohttp2 = Nothing
```

DELAYING SAS CODE

My last example is a simple one. The following code will allow you to submit a program, and the program will remain running in a paused state until the specified time elapses. The time must be given in milliseconds:

```
'SleepandQuit.vbs

WScript.Sleep(12000)
WScript.Quit
```

GOING FURTHER

This is only the beginning of what you can accomplish using SAS and VBScript. The following are examples of other tasks that can be accomplished.

- In an organization where the telephone system utilizes Cisco equipment, scripts can be written to make a phone call under certain circumstance. A programmer might want an alert if an important process errors after business hours.
- Any script that you write or that you have SAS create can be automated by adding the script to the Windows Task Scheduler. This is exactly how SAS Enterprise Guide automates a process. However, keep in mind that the computer you are working on must be running at the time the script is to run. If you want a script to run overnight, but your computer could be restarted for updates, you may need to work with your IT department for a server-based solution.
- VBScript can send keystrokes as if a user was typing.
- VBScript can be used to geocode data for use in SAS maps using the Google or Yahoo geocoding API.

CONCLUSION

If you need to perform a statistical analysis or data manipulation, you can probably do it in SAS. If you need to perform a windows task, you can probably do it with VBScript. More importantly, when you can automate a process programmatically using a combination of these two technologies, why would you ever do it manually?

REFERENCES

ActiveXperts. (n.d.). *Scripts Collection*. Retrieved from ActiveXperts software: http://www.activexperts.com/activmonitor/windowsmanagement/scripts/

Hemedinger, C. (2012). Not Just for Scheduling: Donig More with SAS Enterprise Guide Automation. *SAS Global Forum.* SAS Institute Inc. Retrieved from http://support.sas.com/resources/papers/proceedings12/298-2012.pdf

Microsoft. (2014). *Microsoft Developer Network*. Retrieved from Windows Script Host Basics: http://msdn.microsoft.com/en-us/library/ec0wcxh3(v=vs.84).aspx

Support.SAS.com. (n.d.). SASEGScripting Help. Retrieved from http://support.sas.com/documentation/onlinedoc/guide/SASEGScripting41.zip

W3Schools. (n.d.). *VBScript Tutorial*. Retrieved from W3Schools: http://www.w3schools.com/vbscript/

CONTACT INFORMATION

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

Christopher Johnson
BrickStreet Insurance
400 Quarrier Street
Charleston, WV 25301
(304) 941-1000 Ext. 5359
(304) 941-1186
Christopher.Johnson@BrickStreet.com
www.BrickStreet.com
www.codeitmagazine.com