Paper 411-2013

Using VBA to Debug and Run SAS® Program Interactively, Run Batch Jobs, Automate Output, and Build Applications

Feng Liu, Genworth Financial Inc, Richmond, VA
Ruiwen Zhang, Cary, NC

Abstract

SAS® Enterprise Guide® provides an API that allows users to automate almost every aspect of running Enterprise Guide projects or simply SAS® programs. Visual Basic for Applications (VBA) under Microsoft Excel provides a rich environment for debugging and running VBA applications. This paper shows how you can use VBA to access the automation API of Enterprise Guide to do sophisticated tasks or build your own applications that run SAS. Through VBA, you can create SAS programs on the fly, debug and run programs, analyze SAS listing output, save SAS log to files, and examine SAS ODS. You can accomplish many tasks like running PROC EXPORT automatically, which is not feasible through Enterprise Guide's main interface. Advanced SAS users can run batch jobs, schedule jobs in parallel, or use SAS output as input to other applications.

Introduction

SAS Enterprise Guide offers point-and-click access to all of its feature and greatly improve programming efficiency, especially for non-programmer. In a typical SAS Enterprise Guide environment, we connect to remote Metadata Server (it runs on remote mode - there is no need to print "RSUBMIT" and "ENDRSUBMIT" statement before and after each code), and we do not install local SAS library or PC SAS. Due to the restriction, SAS procedures like PROC EXPORT, PROC IMPORT does not work from UNIX server to local, and SAS EG does not recognize local machines (local and network drives). To use these functions, one needs to go through guided wizard and point-and-click through SAS EG main interface. For example, we generate several SAS datasets on remote server and want to export to local in CSV or Excel format. For each SAS dataset, we need to click "Export" button on SAS EG menu, choose output location(local machine), specify output file format and output file name etc, and click "OK" to proceed. The whole process needs to be repeated for every dataset. There is no SAS code generated for this kind of EG task because EG handle the connection between UNIX and local behind the scene for you. This makes us wonder whether there is a better way to accomplish this task in a programmable fashion (i.e, you can specific the format, filename etc in code and run in batch mode). Installation of a local SAS library as shown in reference[1] is one option, that would require installation of a local SAS instance inside SAS EG and user can toggle between "remote" and "local" mode, when in "local" mode, user can access remote server through "RSUBMIT" statements and at the same time access local drives.

We explore many other approaches like SAS "Prompt Manager" inside SAS EG, and other SAS stored processes. These approaches would eliminate some point-and-clicks, but not all, it still require some clicks. We land on Chris's paper [2][3] about SAS Enterprise Guide API and COM-based object. The SAS Enterprise Guide API provides a powerful mechanism to link SAS remote server to local. The concepts are familiar to us under windows environment, and this motivates us to do more research and we eventually find VBA a great tool to run almost every SAS jobs we needed. We can create SAS programs on the fly, debug and run the programs, analyze/save SAS Listing Output, write complete SAS logs to files, examine SAS Output Delivery System(ODS), email the results or send out notifications. This approach addresses the programming issues of automation of SAS procedures between UNIX(or remote server) and local, and it does not require installation of local SAS library or PC SAS inside SAS EG. The code is also consistent, no "RSUBMIT" and "ENDRSUBMIT" in all SAS code, all SAS code is running on remote server.

We successfully create a self-contained SAS running environment in Excel without running SAS EG main user interface. For SAS end users, knowledge of VB language is not necessary (macro being setup), one can only work in Excel work sheet to write SAS programs. VBA not only provides an easy-to-use tool to run SAS programs similar to SAS command-line tool, but it seamlessly connects other objects/modules for us to

build our own applications that run SAS as well. VBA under Excel is available for almost every windows user. We find VBA useful to run SAS jobs through its step-by-step debug mode: watch intermediate variables, step into, run to cursors. Besides that, we can also access Excel formula, cells, named range, macros, and more. We also use VBA as toolkit to debug SAS codes before developing .NET application with SAS.

Difference Between VBA and VBScript

As noted in Chris's paper, you can create VBScripts using any text editing tool like notepad, and save file as "newprog.vbs" and run the scripts through command-line:

```
C:\scripts> cstript  newprog.vbs
```

In contrast, VBA program are stored inside Excel program through macro-enabled file format xlsm and all driven from Excel macros. They are both "VB" scripts language and largely the same, but there are difference, for example "WScript" is not recognized in VBA. VBA also provides a rich environment for syntax highlighting, syntax checking, auto completion, debugging, accessible to excel formulas or objects.

Write VBA Program to Access SAS

Create an Excel workbook, and save it as macro-enabled file xlsm format. Then click "Alt-F11", it brings you to VBA macro-editing mode. Right-click on the "project" on the left-hand panel, and choose "Insert" -> "Module", you will see a new module "Module1" created under "Modules" folder.

EXAMPLE 1: FIRST VBA PROGRAM

The following code is to create an instance of SAS Enterprise Guide Application Object and loop through all available metadata profiles assigned to current SAS EG user by IT SAS administrator.

```
Option Explicit
Public Application   'Application
Public Project       'Project object
Public sasProgram    'Code object (SAS program)
Public log1 As String

Sub Example1()
Set Application = CreateObject("SASEGObjectModel.Application.4.3")
Dim i As Long
Dim x(0 To 10) As String
    For i = 1 To Application.Profiles.Count - 1
    x(i) = "Profile available: " _
    & Application.Profiles.Item(i).Name _
    & ", Host: " & Application.Profiles.Item(i).HostName _
    & ", Port: " & Application.Profiles.Item(i).Port
    Next
End Sub
```

We declare "Application", "Project" and "sasProgram" all three objects "Public" since the connection will be preserved after each macro run unless the whole VBA project is reset. We show in later examples that we access these objects without the requirement to re-create objects and make explicit re-connections in every macro. This is like SAS command-line, you open one connection to SAS server and type SAS commands as long as your SAS session is active. In this macro, we create a SAS EG object and assigned to "Application". In VBA, you type "_" to do line continuation.

Click "F8" to debug and run the code line-by-line, it will highlight current line-of-code in $Yellow$. In step-by-step mode, you choose "Debug" in the menu, and choose "Add Watch" to observe intermediate values for choosen variable at each step. We add "Application" variable to the watch list, and you could click "+" sign

to expand the tree after running "CreateObject". Full attributes of "Application" object are available to you. Figure 1 shows the "Watches" panel and SAS Enterprise Guide Version is 4.3.0.0.

The next FOR loop is to extract all profiles and their properties for the current SAS EG user. You can also add "x" variable in this example to "Watches" list so that you could browse each profile contents right after running the code. One of the profile names is needed in the following examples to connect to SAS server and run SAS codes.

Note that you might be prompted a windows to enter login credentials if your SAS Enterprise Guide profile requires login, or you could save your login in SAS Enterprise Guide profiles.

Run SAS Programs Interactively

This is an interesting topic. The Step-by-Step running mode of VBA allows us to run SAS code piece-by-piece as needed, view outputs like traditional SAS listing output and SAS LOG at each step. The complete project listings and logs (not individual piece-by-piece log) could be appended and stored all together in Excel or into external files. For SAS end users, knowledge of VB language is not necessary, one can work only in Excel worksheet to write SAS programs, see EXAMPLE 4.

EXAMPLE 2: RUN A SAS PROGRAM

The following example makes a connection to a SAS server, runs a SAS program, and displays SAS "LOG" to the user while also saving to local file, it is written in one VBA macro.

```
Sub Example2()
' Set to your metadata profile name
Application.SetActiveProfile ("SAS BI")

' Create a new Project
Set Project = Application.New
' Add a new code object to the Project
Set sasProgram = Project.CodeCollection.Add
' set the results types, overriding Application defaults
sasProgram.UseApplicationOptions = False
sasProgram.GenListing = True
sasProgram.GenSasReport = False

' Set the server
sasProgram.Server = "Risk"

' Get current workbook path
Dim DatafileDir As String
DatafileDir = ThisWorkbook.Path

' Create the SAS program to run
sasProgram.Text = "libname data1 " & Chr(34) & _
"/sas/risk/analysis/data" & Chr(34) & ";" & _
"data work.bmtvar; set data1.bmtvar; run;"

' Run the code
sasProgram.Run

' Save the log file to LOCAL disk
sasProgram.Log.SaveAs _
DatafileDir & "\" & "script1.log"
log1 = sasProgram.Log.Text
MsgBox sasProgram.Log.Text
```

```
End Sub
```

The "SetActiveProfile" uses one of the profile names obtained in previous example. The "Applicatioin.New" and "CodeCollection.Add" create a project and add a new program code to it. Application-level options for ODS statements are overridden.

The SAS program is created by assigning plain text to “sasProgram.Text", the code is running on one of the server “Risk" from servers list. Note that the double quote needs to be replaced by “Chr(34)" in VBA language. The LOG is saved to local file called “script1.log" and also displayed on a pop-up message window as shown in Figure 2.

Figure 3 shows many useful information like SAS outputs of “LIST", “LOG", and “OUTPUTDATASETS" in the "Watches" panel, the SAS codes are ran on the remote server and results are pushed back to COM-object and available in Excel. One can explore it and decide what to do with the output.

EXAMPLE 3: RUN SAS CODES PIECE-BY-PIECE INTERACTIVELY

The following code demonstrates how to run subsequent SAS programs piece-by-piece. It is not required to create new project or create new program object in subsequent macros since we can re-use the program object. EXAMPLE 3 can be executed immediately after Example 2 is run. The SAS log could be viewed through message windows after running each piece, therefore decision to make/change new code could be dependent on previous results, which works in an interactive fashion.

```
Sub Example3()
sasProgram.Text = "libname data2 " & Chr(34) & _
"/sas/risk/analysis/data1" & Chr(34) & ";" & _
"data work.bmtvar; set data2.certs(obs=10); run;"
sasProgram.Run
log1 = log1 & sasProgram.Log.Text
MsgBox sasProgram.Log.Text
End Sub
```

The debug mode in Excel is powerful and provides options like "Step Into", "Step Over", "Run to Cursor" etc so that we can trace complex process flows. We could also reference external data sources like Excel Cell values as input variable to generate dynamic SAS code.

EXAMPLE 4: WRITE SAS CODES IN EXCEL WORKSHEET

EXAMPLE 4 runs the same SAS code as in EXAMPLE 3 while SAS code is written in EXCEL “front end" worksheet instead of VBA macros, in this example Cell “B1", it also brings back SAS Log to Cell "B3". Notice that SAS code in EXAMPLE 4 is exactly same as traditional SAS programs in PC SAS or SAS EG, no conversions like double quotes to “Chr(34)" and no line continuation. One can copy SAS codes (one statement or multiple statements) from existing SAS programs and run. Users can interface through Excel Worksheet only to write and debug SAS programs. SAS Logs including error messages will be brought back to Excel worksheet right after running, no need to open another log file in a typical SAS UNIX command-line environment. Meanwhile, users can append every SAS Logs you are debugging ("Log1" in this example) to a complete LOG file without concerns about LOGs being overwritten by new run.

```
'SAS code in Excel Worksheet Cell "B1", same as traditional SAS programs;
libname data2 "/sas/risk/analysis/data1";
data work.bmtvar;
set data2.certs(obs=10);
run;
```

```
Sub Example4()
sasProgram.Text = Range("B1")
sasProgram.Run
```

```
log1 = log1 & sasProgram.Log.Text
Range("B3") = log1
End Sub
```

Figure 1 Run macro line-by-line, "Add Watches" to show variable attributes in "Watches" panel

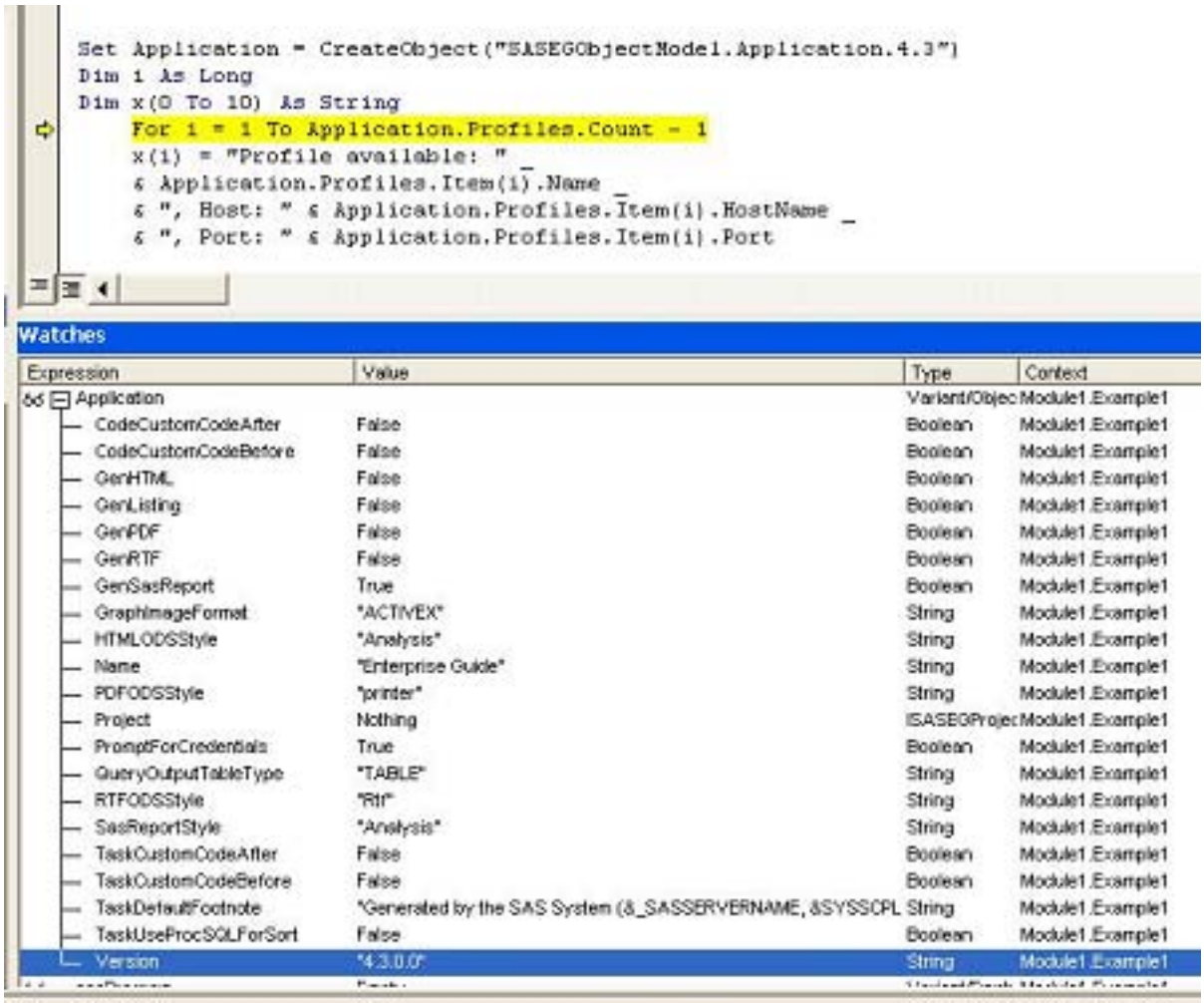


Figure 2 SAS Log displayed in Excel

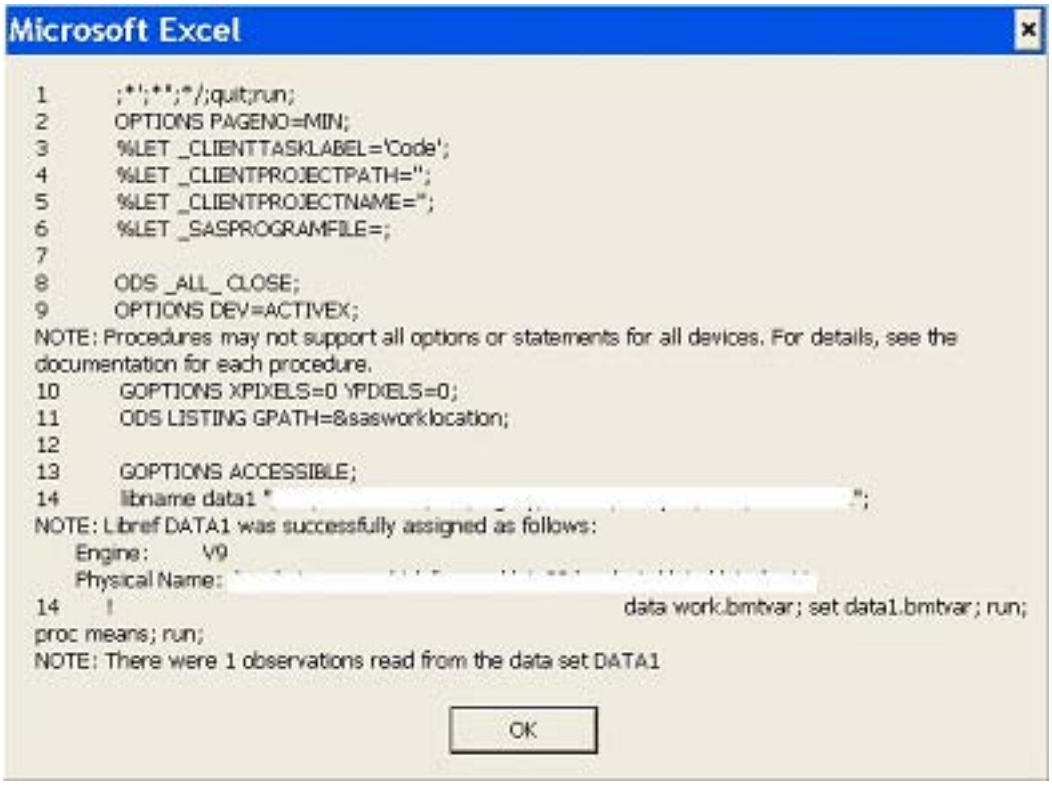


Figure 3 SAS code outputs like "LIST", "LOG", "OUTPUTDATESETS" are available in "Watches" panel

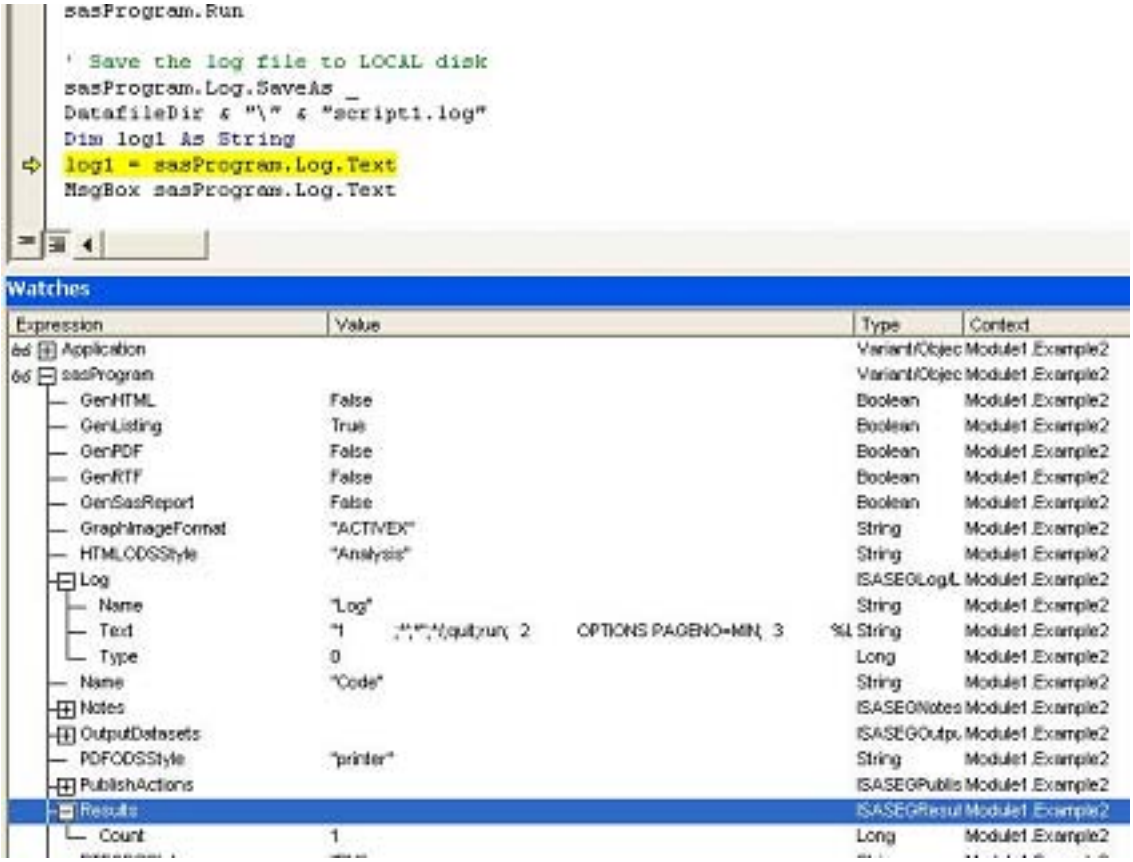


Figure 4 SAS automation output: Datasets, Listing, and Logs

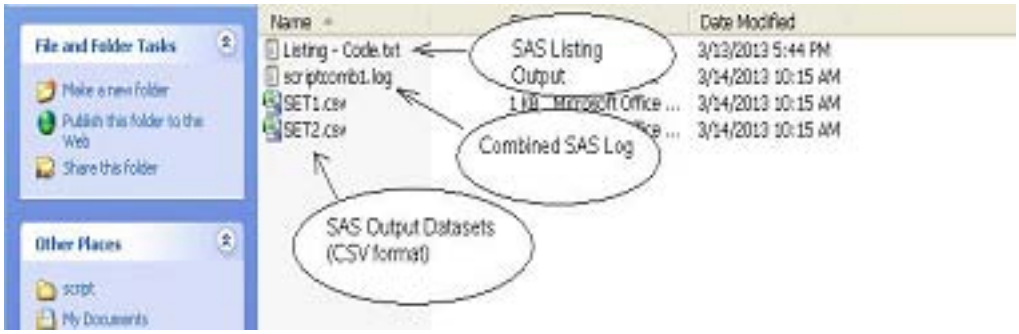
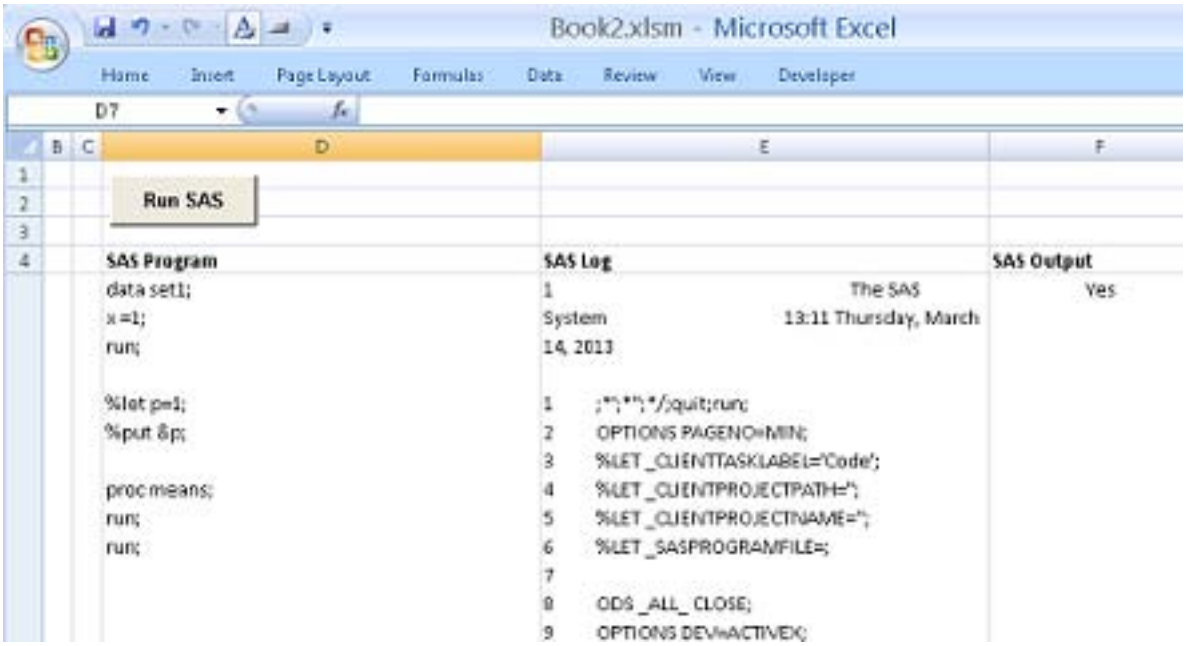


Figure 5 A VBA application that runs SAS



Automate SAS Output and PROC EXPORT

SAS Enterprise Guide API provides a powerful mechanism to link SAS remote server to local machine without local SAS library (or "local" mode). The main issues of non-feasibility of "PROC EXPORT" between UNIX and Local could be addressed and point-and-click version of "Export" in SAS EG main is transformed to programs and ran in batch mode. Many other SAS tasks can be programmed and ran automatically through members and methods of SAS Enterprise Guide API (in SASEGScripting.dll), see scripting reference.

EXAMPLE 5: SAS OUTPUT AUTOMATION (DATASET, LISTING, LOG)

The following example shows how to do "PROC EXPORT" automatically. This example illustrates three type of typical SAS outputs: SAS datasets (either SAS dataset or CSV files), SAS listing output (ex. listing table from PROC MEANS), and SAS Logs.

```
Sub Example5()
sasProgram.Text = "libname data2 " & Chr(34) & _
"/sas/risk/analysis/data1" & Chr(34) & ";" & _
"data work.set1; set data2.certs(obs=20); run;" & _
"data work.set2; set data2.ctrs(obs=20);  run; proc means; run;"
sasProgram.Run
```

```
MsgBox ("program run successfully!")

' Get current workbook path
Dim DatafileDir As String
DatafileDir = ThisWorkbook.Path

' Export SAS dataset to local CSV files
Dim n as Integer
Dim dataName
For n = 0 To (sasProgram.OutputDatasets.Count - 1)
dataName = sasProgram.OutputDatasets.Item(n).Name
sasProgram.OutputDatasets.Item(n).SaveAs _
DatafileDir & "\" & dataName & ".csv"
Next

' Save SAS listing output to TXT files
For n = 0 To (sasProgram.Results.Count - 1)
dataName = sasProgram.Results.Item(n).Name
sasProgram.Results.Item(n).SaveAs _
DatafileDir & "\" & dataName & ".lst"
Next

' Save complete SAS logs to file
log1 = log1 & sasProgram.Log.Text
MsgBox sasProgram.Log.Text

Dim sFName As String
sFName = DatafileDir & "\" & "scriptcomb1.log"
'Get an unused file number
Dim intFNumber As Integer
intFNumber = FreeFile
'Create a new file (or overwrite an existing one)
Open sFName For Output As #intFNumber
Write #intFNumber, log1
'Close the text file
Close #intFNumber
End Sub
```

Assume we have SAS datasets at server locations. The “export" technique is to copy datasets from server location to “work" library through "staging" process. SAS EG API provides reference to output datasets in work library, which can then be exported to either .csv, .xls format or SAS dataset on local machines. In this example, two CSV files "set1.csv" and set2.csv" are generated in Figure 4. This program also saves SAS listing output as well as SAS logs to local text files.

Build a VBA application that runs SAS

With all the bells and whistles, one can create a sophisticated VBA application that runs SAS. Figure 5 shows a SAS running application, type SAS programs or copy and paste your own programs(including SAS macros), click “Run SAS", the SAS program runs on remote server and generates logs, listing and output datasets at local machine. It is self-contained and no need to open SAS EG main application.

Conclusion

This paper presents ideas of using VBA to debug and run SAS codes through SAS Enterprise Guide API. VBA provides a powerful running environment and debug tool which can run SAS programs smoothly and efficiently as a SAS client does, but also seamlessly link remote server to local so that we could automate SAS batch jobs without SAS local mode. We could also access local applications and objects.

Reference

Using SAS Enterprise Guide the Same Way as Base SAS and More Rahman Sarker, Royal Bank of Canada, Toronto, ON Paper 300-2012 SAS GLOBAL FORUM 2012

Boost Your Programming Productivity with SAS Enterprise Guide Chris Hemedinger, SAS Institute Inc, Cary, NC Paper 019-30 SUGI 30

Not Just for Scheduling: Doing More with SAS Enterprise Guide Automation Chris Hemedinger, SAS Institute Inc, Cary, NC Paper 298-2012 SAS GLOBAL FORUM 2012

SAS Enterprise Guide Scripts "http://support.sas.com/documentation/onlinedoc/guide/"

CONTACT INFORMATION

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

Feng Liu
Genworth Financial Inc.
Feng.Liu@genworth.com