Paper SM01

PowerS(h)elling SAS

Tom Van Campen, SGS Life Science Services, Mechelen, Belgium
Benny Haemhouts, SGS Life Science Services, Mechelen, Belgium

ABSTRACT

In the search to facilitate batch processes in SAS®, we came across PowerShell™. PowerShell is a product from Microsoft® which is mainly used by System Administrators to automate Operating System tasks.

After following several online tutorials, we were able to quickly adopt the PowerShell language. We also learned that PowerShell could be used to create a graphical user interface (GUI) for configuration and control of SAS processes.

Non-SAS profiles can now run certain SAS related tasks themselves. They will just need to select the appropriate parameters in the GUI. The collected information is then passed through to the SAS batch process.

The GUI can also guide experienced SAS users through certain procedures. This reduces the cumbersome task of referencing manuals and flowcharts.

As such, the power of SAS can be promoted to a broader audience.

Nowadays, SGS has several PowerShell interfaces available and we would like to share our experiences.

INTRODUCTION

WHAT IS POWERSHELL

Windows® PowerShell (further referenced as PowerShell) is a scripting language from Microsoft which is mainly used to automate Operating System tasks. PowerShell comes installed by default in every Windows, starting from Windows 7 SP1.

The Integrated Scripting Environment (ISE) is the preferred application to develop your tools.

PowerShell information is abundantly available on the internet. This reduces the learning curve to adopt the language. A possible starting point would be to follow some tutorials provided by Microsoft (see appendix).

It is also possible to create a graphical user interface (GUI) using PowerShell.

Windows PowerShell v5.1 was used for the examples in this paper.

INTERACTION WITH SAS

PowerShell can launch programs via the command line.
This enables us to batch SAS programs and use the -SYSPARM functionality already available in SAS.
-SYSPARM allows to pass through information from the command line to the SAS environment.
On the other end, SAS can provide a return code when it finishes.

The actual processing will still be done in the validated SAS environment.

ADVANTAGES OF THE GUI

A GUI provides a more intuitive way to collect the required information.
Rather than adapting a SAS program, information can be gathered using text boxes, check boxes, …

In addition, a GUI can provide the user with visual guidance on the run order of multiple programs.
Using the return code from SAS, the GUI can control the process by enabling/disabling the next step.

A GUI also allows a non-SAS profile to configure and execute certain SAS batches.
It is important to foresee the necessary error handling and log checking.
In this way, the non-SAS profile will be made aware when to contact the developer.

POWERSHELL GUI BASICS

To try out the below examples, we advise to copy/paste the code in your Windows PowerShell ISE.

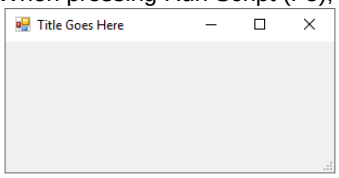
CREATE EMPTY FORM

This is the code to create/show an empty form:

```
# Define Form
Add-Type -AssemblyName System.Windows.Forms
$frmSAScontroller               = New-Object system.Windows.Forms.Form
$frmSAScontroller.Text          = 'Title Goes Here'
$frmSAScontroller.StartPosition = 'CenterScreen'
$frmSAScontroller.Width         = 320
$frmSAScontroller.Height        = 160

# Display Form
$frmSAScontroller.ShowDialog()
```

➔ This code will need to be saved as a .ps1 file prior to execution.

When pressing Run Script (F5), following output is shown:



Remarks:
The line "`Add-Type -AssemblyName System.Windows.Forms`" will load the needed GUI components for creating forms.
During an ISE session, these components are automatically loaded. When launched outside of the ISE these components are not pre-loaded.

LAUNCHING THE FORM OUTSIDE ISE

End users will not be opening and running the .ps1 through ISE.
They will use an alternative method which immediately runs the script:

- right mouse button - "Run with PowerShell"
- launch directly via command line or .bat file
  tip: This method enables you to pass specific information to a central PowerShell script

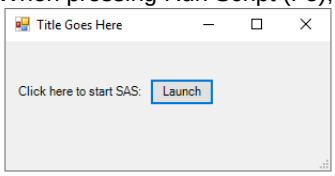
This will create the exact same form as shown above.

CREATE BASIC FORM

Below code can be used to create a basic form with a button and a label:

```
# Define Form
Add-Type -AssemblyName System.Windows.Forms
$frmSAScontroller               = New-Object system.Windows.Forms.Form
$frmSAScontroller.Text          = 'Title Goes Here'
$frmSAScontroller.StartPosition = 'CenterScreen'
$frmSAScontroller.Width         = 320
$frmSAScontroller.Height        = 160
$controls=$frmSAScontroller.Controls

# Add Label
$lblStartSAS          = New-Object System.Windows.Forms.Label
$lblStartSAS.Name     = 'lblStartSAS'
$lblStartSAS.Location = New-Object Drawing.Point 10, 40
$lblStartSAS.Text     = 'Click here to start SAS:'
$lblStartSAS.AutoSize = $true
$controls.Add($lblStartSAS)

# Add Button
$cmdStartSAS          = New-Object Windows.Forms.Button
$cmdStartSAS.Name     = 'cmdStartSAS'
$cmdStartSAS.Text     = 'Launch'
$cmdStartSAS.Size     = New-Object System.Drawing.Size(60, 25)
$cmdStartSAS.Location = New-Object Drawing.Point 135, 35
$controls.Add($cmdStartSAS)

# Display Form
$frmSAScontroller.ShowDialog()
```

When pressing Run Script (F5), following output is shown:



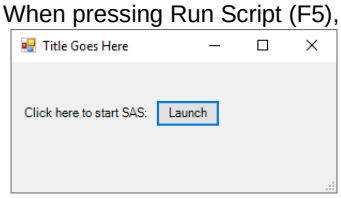
Remarks:
The line "`$controls=$frmSAScontroller.Controls`" will provide an alias which can be used to place components on the form using "`$controls.Add($xxx)`".

ADD AN ACTION TO THE BUTTON

Upon clicking the button, a SAS batch is initiated:

```
# SAS Locations
$sasExecutable = 'C:\Program Files\SASHome\x86\SASFoundation\9.4\SAS.EXE'
$sasConfigFile = 'C:\Program Files\SASHome\x86\SASFoundation\9.4\sasv9.cfg'

# Define Form
Add-Type -AssemblyName System.Windows.Forms
$frmSAScontroller               = New-Object system.Windows.Forms.Form
$frmSAScontroller.Text          = "Title Goes Here"
$frmSAScontroller.StartPosition = 'CenterScreen'
$frmSAScontroller.Width         = 320
$frmSAScontroller.Height        = 160
$controls=$frmSAScontroller.Controls

# Add Label
$lblStartSAS          = New-Object System.Windows.Forms.Label
$lblStartSAS.Name     = 'lblStartSAS'
$lblStartSAS.Location = New-Object Drawing.Point 10, 40
$lblStartSAS.Text     = 'Click here to start SAS:'
$lblStartSAS.AutoSize = $true
$controls.Add($lblStartSAS)

# Add Button
$cmdStartSAS          = New-Object Windows.Forms.Button
$cmdStartSAS.Name     = 'cmdStartSAS'
$cmdStartSAS.Text     = 'Launch'
$cmdStartSAS.Size     = New-Object System.Drawing.Size(60, 25)
$cmdStartSAS.Location = New-Object Drawing.Point 135, 35
$controls.Add($cmdStartSAS)

# Add SAS Batch
$controls['cmdStartSAS'].Add_Click({
   Write-Host "Launching SAS"
   $sasProgram = 'X:\location\program_name'
   & "$sasExecutable" "$sasProgram.sas" -NOLOGO -RSASUSER -CONFIG "$sasConfigFile"
     -LOG "$sasProgram.log" | Out-Null
   if ($lastexitcode -eq 0){
      $controls['cmdStartSAS'].BackColor = 'LightGreen'
   } else {
      $controls['cmdStartSAS'].BackColor = 'Red'
   }
})

# Display Form
$frmSAScontroller.ShowDialog()
```

When pressing Run Script (F5), following output is shown:



- ➔ upon pressing launch, the SAS batch will initiate.
- ➔ after completion, the button will turn green/red depending on the SAS return code (via `$lastexitcode`). tip: use "`%abort abend;`" in SAS to indicate that the run failed.

Remarks:
The "`Write-Host`" will write the given text to the PowerShell log.
The "`&`" will execute SAS from the command line.
The "`| Out-Null`" forces PowerShell to wait until process completion.

USE OF FUNCTIONS AND SYSPARM

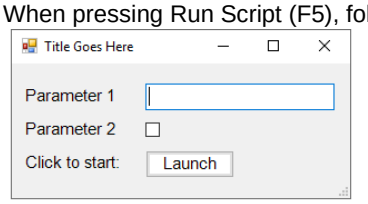
The first section shows how to influence the font and introduces functions to improve code readability:

```
# SAS Locations
# -------------
$sasExecutable = 'C:\Program Files\SASHome\x86\SASFoundation\9.4\SAS.EXE'
$sasConfigFile = 'C:\Program Files\SASHome\x86\SASFoundation\9.4\sasv9.cfg'

# Define Form
# -----------
Add-Type -AssemblyName System.Windows.Forms
$frmSAScontroller               = New-Object system.Windows.Forms.Form
$frmSAScontroller.Text          = "Title Goes Here"
$frmSAScontroller.StartPosition = 'CenterScreen'
$frmSAScontroller.Width         = 320
$frmSAScontroller.Height        = 160
$controls=$frmSAScontroller.Controls

# Define fonts
# ------------
$FontBold    = New-Object System.Drawing.Font('Arial', 11, [System.Drawing.FontStyle]::bold)
$FontItalic  = New-Object System.Drawing.Font('Arial', 11, [System.Drawing.FontStyle]::italic)
$FontRegular = New-Object System.Drawing.Font('Arial', 11, [System.Drawing.FontStyle]::regular)

# Functions
# ---------
Function CreateLabel ($name_, $locX_, $locY_, $text_, $font_=$FontRegular) {
   $tmpLbl           = New-Object System.Windows.Forms.Label
   $tmpLbl.Name      = $name_
   $tmpLbl.Location  = New-Object Drawing.Point $locX_, $locY_
   $tmpLbl.Text      = $text_
   $tmpLbl.AutoSize  = $true
   $tmpLbl.Font      = $font_
   $tmpLbl.ForeColor = 'Black'
   $tmpLbl.BackColor = [System.Drawing.Color]::FromName("Transparent")
   $controls.Add($tmpLbl)
}

Function CreateTextbox ($name_, $locX_, $locY_, $sizeX_, $sizeY_, $text_, $font_=$FontRegular) {
   $tmpTxt           = New-Object System.Windows.Forms.TextBox
   $tmpTxt.Name      = $name_
   $tmpTxt.Location  = New-Object Drawing.Point $locX_, $locY_
   $tmpTxt.Size      = New-Object System.Drawing.Size($sizeX_, $sizeY_)
   $tmpTxt.Text      = $text_
   $tmpTxt.Font      = $font_
   $controls.Add($tmpTxt)
}

Function CreateButton ($name_, $locX_, $locY_, $sizeX_, $sizeY_, $text_) {
   $tmpCmd           = New-Object Windows.Forms.Button
   $tmpCmd.Name      = $name_
   $tmpCmd.Text      = $text_
   $tmpCmd.Size      = New-Object System.Drawing.Size($sizeX_, $sizeY_)
   $tmpCmd.Location  = New-Object Drawing.Point $locX_, $locY_
   $tmpCmd.BackColor = 'White'
   $tmpCmd.ForeColor = 'Black'
   $tmpCmd.Font      = $fontRegular
   $controls.Add($tmpCmd)
}

function CreateCheckbox ($name_, $locX_, $locY_, $sizeX_, $sizeY_, $text_) {
   $tmpChk           = New-Object Windows.Forms.checkbox
   $tmpChk.Name      = $name_
   $tmpChk.Text      = $text_
   $tmpChk.Size      = New-Object System.Drawing.Size($sizeX_, $sizeY_)
   $tmpChk.Location  = New-Object Drawing.Point $locX_, $locY_
   $tmpChk.ForeColor = 'Black'
   $tmpChk.Checked   = $false
   $controls.Add($tmpChk)
}

# Add Components
# --------------
CreateLabel    lblParam1      10  20   'Parameter 1'
CreateLabel    lblParam2      10  50   'Parameter 2'
CreateLabel    lblStart       10  80   'Click to start:'

CreateTextbox  txtParam1     120  18   170  25   ''
CreateCheckbox chkParam2     120  48    25  25   ''
CreateButton   cmdStartSAS   120  78    80  25   'Launch'
```

The second section shows how the collected information is passed on to SAS using -SYSPARM:

```
# Add SAS Batch
# -------------
$controls['cmdStartSAS'].Add_Click({
   $sasProgram    = 'F:\MACROSSAS_V09.4M5\01_DEV\0. admin\0. Powershell\program'
   Write-Host "Launching SAS"
   $varSysparm = $controls['txtParam1'].Text + '|' + $controls['chkParam2'].Checked
   & "$sasExecutable" "$sasProgram.sas" -NOLOGO -RSASUSER -CONFIG "$sasConfigFile"
     -LOG "$sasProgram.log" -SYSPARM "$varSysparm" | Out-Null
   if ($lastexitcode -eq 0){
      $controls['cmdStartSAS'].BackColor = 'LightGreen'
   } else {
      $controls['cmdStartSAS'].BackColor = 'Red'
   }
})

# Display Form
# ------------
$frmSAScontroller.ShowDialog()
```

When pressing Run Script (F5), following output is shown:



Remarks:
There is only one -SYSPARM. If you would like to pass multiple parameters to SAS, a delimiter has to be used to parse the information in the SAS session.

LESSONS LEARNED

As SAS programmers, we noticed that PowerShell has some similarities with SAS.
This allows to quickly adopt this language.

Some examples are:

- Functions in PowerShell and macros in SAS:
  Both allow you to reduce your code and make coding more dynamic using parameters.
- Variables in PowerShell ($var) and macro variables in SAS (&var):
  PowerShell variables can contain either text or objects and are thus more extensive.
  When using them as text, they function similar to a macro variable in SAS.
  Note: double and single quoting have a similar effect.
  Meaning, PowerShell variables do not resolve between single quotes.

The GUI examples shown above are merely an introduction. Some other features are:

- list boxes
- reading input files

Note that PowerShell GUI designers are available online. They could serve as inspiration.

CONCLUSION

We found that PowerShell is very useful to facilitate the execution of certain SAS processes.
Both SAS and non-SAS profiles can easily control the information that needs to be passed on to the SAS session.
In addition, the GUI allows to visualize the process flow in a more organized way.
This reduces the risk of human error and increases efficiency.

In this strategy, PowerShell is only used as an interface on SAS. This means that the validated process remains in the SAS environment.
The usage of -SYSPARM allowed us to pass information to SAS rather than altering the program code each time.
This also allows to place certain codes in a central read-only environment for the user.

Given the good reception of the first PowerShell tool by our users, we have created several PowerShell GUIs for our SAS processes.

APPENDIX:

Session 1, Getting Started with Microsoft PowerShell (more focused on admin role):
https://channel9.msdn.com/Series/Getting-Started-with-Microsoft-PowerShell
Session 2, Advanced Tools & Scripting with PowerShell 3.0 Jump Start (more focused on GUI design):
https://channel9.msdn.com/Series/Advanced-Tools-and-Scripting-with-PowerShell-3.0-Jump-Start

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:
clinicalresearch@sgs.com
EUROPE: +32 15 27 32 45
AMERICAS: + 877 677 2667
WWW.SGS.COM/CRO