Paper 068-30

A Hacker's Guide to SAS® Environment Customization

Brian Fairfield-Carter, PRA International, Victoria, BC (Canada)
Stephen Hunt, PRA International, Victoria, BC (Canada)

ABSTRACT

Most SAS users have at least a few customizations and personalization they like to make to their SAS environments. Real hackers, naturally, like to explore the extremes to which these customizations can be taken, mostly to shroud themselves in mystery, but partly to try and discover practical and useful tricks and tools. In this spirit, four customizations as presented here:

1. A small dynamic HTML application that makes copies of your user profile, and sets configuration option "-SASUSER" to point to a free copy each time a new SAS session is launched (allowing multiple concurrent sessions to all support the same preferences and options)
2. A SAS enhanced editor button configured to plug meta-data into a PROC REPORT "code template"
3. Using the SAS enhanced editor to write and test programs in 'foreign languages' (in both interpreted and compiled environments)
4. Dynamically setting the "-SASINITIALFOLDER" configuration option

Through these examples, the use of SAS in conjunction with Windows Scripting Technologies, the Windows command interface, and the GNU Compiler Collection is demonstrated, and will hopefully serve as a starting point for the reader's own hacking.

INTRODUCTION

When writing an application for your own use, since you are free to make assumptions about your own knowledge and understanding, you can get away with a lot that simply wouldn't fly when writing an application for other users. This is why these 'hacker' applications are the most fun to write, and also why they are the most likely to promote an aura of eccentricity. This is also why they are the perfect place to control customization to your own SAS programming environment.

Typical customizations are set in one of three ways: through an 'options' statement in a SAS program, by setting preferences and short-cut keys, and creating key-board macros in the SAS interface (these are stored in a user profile catalog and associated auxiliary files), or by passing configuration options to the SAS executable file at start-up (some, but not all of these options are stored in the SASV8.CFG file). For a hacker, it's really the latter 2 of these that present interesting challenges: your customized profile catalog can only be referenced by a single SAS session, and SAS configuration options can only be set at start-up; this means that SAS itself can not be used to modify these, and some external environment must be employed.

PROPAGATING YOUR USER PROFILE, FOR RUNNING MULTIPLE CUSTOMIZED SESSIONS

SAS keyboard shortcuts (i.e. to clear the log and output windows and submit programs), and options/preferences settings (i.e. to activate or suppress the display of ODS output as it is generated) are stored partially in a user profile, in the form of a SAS System Catalog, and partially in a collection of auxiliary files called SAS System "Item Stores". These files can only be referenced by a single SAS session. This means that when you launch additional concurrent sessions, you will see a note like this in the log:

```
NOTE: Unable to open SASUSER.PROFILE. WORK.PROFILE will be opened instead.
```

Any preferences, options, and other customizations that you have set up will be inoperable in these additional sessions.

A solution is to make copies of the necessary files, and modify the "SASV8.CFG" configuration file to point to these copies before launching a new session. Obviously this would be a tiresome task to attempt by hand, but automation using SAS wouldn't work, since by the time an "x 'copy'" command was submitted, the files would already be locked against copying. (You'd get an error box stating "Cannot copy profile: It is being used by another person or program.")

Dynamic HTML (dHTML) is probably the simplest environment in which to implement an application to carry out these tasks. Even without knowing anything about the language the script components of a dHTML page are written in, you can probably read through the source code and discern more or less what is going on. The example given here consists of a single button:

```
<INPUT TYPE="Button" NAME="LaunchSAS" VALUE="New SAS Session">
```

, which displays in Internet Explorer as

A simple “OnClick” event script (written in Visual Basic Scripting Edition (VBScript)) is provided for the button, to provide a response each time the button is clicked:

```
<SCRIPT FOR="LaunchSAS" EVENT="OnClick">
   nSessions=nSessions + 1
   NewSAS()
</SCRIPT>
```

This event script does two things: it increments a counter (a global variable) which stores the number of SAS sessions that have been launched by the application, and it calls a user-defined function called “NewSAS” to carry out some file handling operations and launch a new SAS session. The function “NewSAS” is declared in a ‘start-up’ script, that runs when the dHTML page loads, and which also initializes “nSessions” as a global variable:

Opening script tag:

```
<SCRIPT LANGUAGE="VBScript">
```

Declaration and initialization of global:

```
Dim nSessions
nSessions=0
```

Start of function declaration:

```
Function NewSAS()
```

Declaration of ‘FileSystemObject’, ‘Network’, and ‘Shell’ objects, for file handling, retrieval of user name, and process launching, respectively:

```
Dim fso_, net, wshshl
Set fso_=CreateObject("Scripting.FileSystemObject")
Set net = CreateObject("WScript.Network")
Set wshshl=CreateObject("WScript.Shell")
```

Check to see if a numbered 'altprofile' folder already exists corresponding to the current iteration; if it doesn’t, create it:

```
If fso_.FolderExists("C:\Documents and Settings\" & net.username & "\My
 Documents\My SAS Files\V8\altprofile" & nSessions)=False Then
fso_.CreateFolder "C:\Documents and Settings\" & net.username & "\My Documents\My
 SAS Files\V8\altprofile" & nSessions
End If
```

Copy over the current version of the profile catalog, and various auxiliary files into this numbered ‘altprofile’ folder (note that a loop is used to prevent the file copying operations from proceeding until the folder exists):

```
a=False
Do Until a=True
 a=fso_.FolderExists("C:\Documents and Settings\" & net.username & "\My
   Documents\My SAS Files\V8\altprofile" & nSessions)
 If a=True Then
 fso_.CopyFile "C:\Documents and Settings\" & net.username & "\My Documents\My
  SAS Files\V8\profile.sas7bcat", "C:\Documents and Settings\" & net.username &
  "\My Documents\My SAS Files\V8\altprofile" & nSessions & "\profile.sas7bcat"
 fso_.CopyFile "C:\Documents and Settings\" & net.username & "\My Documents\My
  SAS Files\V8\templat.sas7bitm", "C:\Documents and Settings\" & net.username &
  "\My Documents\My SAS Files\V8\altprofile" & nSessions & "\templat.sas7bitm"
 fso_.CopyFile "C:\Documents and Settings\" & net.username & "\My Documents\My
 SAS Files\V8\regstry.sas7bitm", "C:\Documents and Settings\" & net.username &
 "\My Documents\My SAS Files\V8\altprofile" & nSessions & "\regstry.sas7bitm"
fso_.CopyFile "C:\Documents and Settings\" & net.username & "\My Documents\My SAS
 Files\V8\sasmbc.sas7bdat", "C:\Documents and Settings\" & net.username & "\My
```

```
 Documents\My SAS Files\V8\altprofile" & nSessions & "\sasmbc.sas7bdat"
fso_.CopyFile "C:\Documents and Settings\" & net.username & "\My Documents\My SAS
 Files\V8\sasmbc.sas7bndx", "C:\Documents and Settings\" & net.username & "\My
 Documents\My SAS Files\V8\altprofile" & nSessions & "\sasmbc.sas7bndx"
fso_.CopyFile "C:\Program Files\SAS Institute\SAS\V8\Sasv8.cfg",
 "C:\Program Files\SAS Institute\SAS\V8\Sasv8_SOURCE.cfg"
End If
Loop
```

Edit the SASV8.CFG file to point to the altprofile<n> folder (note that the source file is a copy of the configuration file, made in the previous step):

```
Dim InText, OutText
Set InText=fso_.OpenTextFile("C:\Program Files\SAS
 Institute\SAS\V8\Sasv8_SOURCE.cfg" ,1)  '1=read
Set OutText=fso_.CreateTextFile("C:\Program Files\SAS Institute\SAS\V8\Sasv8.cfg"
 ,2) '2=write

While Not InText.AtEndOfStream
 string_=InText.ReadLine & NewLine
 If InStr(string_,"-SASUSER " & Chr(34) & "?CSIDL_PERSONAL\My SAS Files\V8" &
  Chr(34))>0 Then
 string_="-SASUSER " & Chr(34) & "?CSIDL_PERSONAL\My SAS Files\V8\altprofile" &
  nSessions & Chr(34)
 End If
 OutText.Write(string_)
 OutText.WriteLine("")
Wend

InText.Close
OutText.Close
```

Launch a new SAS session. This session will use the user profile copy sitting in 'altprofile<n>' because of the alteration made to SASV8.CFG:

```
wshshl.run Chr(34) & "C:\Program Files\SAS Institute\SAS\V8\SAS.EXE" & Chr(34)
```

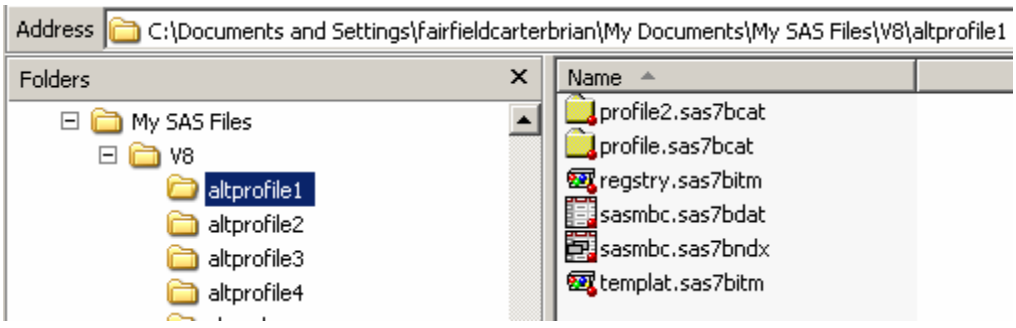
End of function declaration:

```
End Function
```

Closing script tag:

```
</SCRIPT>
```

To re-cap, the script creates a sub-directory under the directory containing the user profile catalog, and copies over all the necessary files:



It then edits the "-SASUSER" line in "SASV8.CFG" to point to this directory:

```
/* Setup the default SAS System user profile folder            */
-SASUSER "?CSIDL_PERSONAL\My SAS Files\V8\ALTPROFILE1"
```

Of course, in so doing it happily ignores this instruction:

```
/* DO NOT EDIT BELOW THIS LINE - INSTALL Application edits below this line */
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

The application exists as a simple text file, with a ".htm" file extension to associate it with Internet Explorer. Each button click creates a new SAS session that supports all the same customizations as the original session.

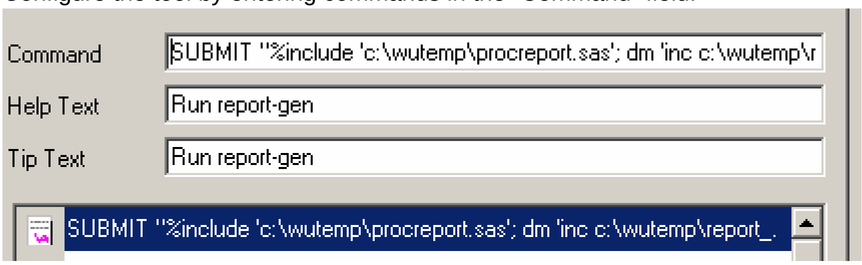
POPULATING A "CODE TEMPLATE" WITH META-DATA

When writing PROC REPORT statements (or virtually any PROC statements, for that matter), the biggest headache is often in correctly transcribing variable names. PROC REPORT can additionally require extensive typing to produce all the necessary "DEFINE" statements. While keyboard macros can be set up to automatically insert procedure "templates", the programmer must still supply all the detail. A large part of the "detail", however, is sitting in SAS metadata tables in the "SASHELP" library, so the challenge is to capture this data and use it to dynamically generate procedure statements that reasonably approximate what you would have typed manually.

ADDING ENHANCED EDITOR TOOLS

In order to create a truly convenient tool, we want to be able to call it from an enhanced editor button. The steps for creating a new button are as follows:

1. First, right-click on the tool bar and select "customize"
2. In the dialog box that pops up, select the "customize" tab, and add a new tool:
3. Configure the tool by entering commands in the "Command" field:



4. Double-click on the entry in the scrolling list, and select an icon for your button:

The new button will now appear in the tool bar, and will execute the command you've supplied when clicked.

DYNAMICALLY GENERATING PROC REPORT STATEMENTS

"Meta-data" is data about data. In other words, for a SAS dataset, it consists of such things as the variables names and attributes (data type, format, label, etc.). The macro below captures meta-data from the dictionary.columns table for the last dataset created in the WORK library, and uses this to generate PROC REPORT statements. Some simple rules are supplied to control column widths, and to deal with missing attribute information.

```
%macro procreport;
proc sql noprint;
```

First, get the name of the last dataset added to the WORK library:

```
   select trim(left(memname)) into :dsetnm from
    (select libname, memname, memtype, crdate, modate from sashelp.vtable
      where libname="WORK" and memtype="DATA" and memname ne "ATTRB"
      group by libname having modate=max(modate));
```

Next, create lists of variables and attributes for this dataset:

```
    select count(*), name, length,
           case when label^="" then compress(label,"'") else "none" end,
           case when format^="" then format else "none" end, type, varnum into
     :nvars_,
     :name_ separated by "~",
     :length_ separated by "~",
     :label_ separated by "~",
     :format_ separated by "~",
     :type_ separated by "~",
     :varnum_ separated by "~"
     from dictionary.columns where libname="WORK" and memname="&dsetnm";
quit;
```

Next, write proc report statements to a text file; if the variable's label is missing, use the variable name as the column heading:

```
data _null_;
  file "c:\wutemp\report_.sas";
  put "PROC REPORT DATA=&dsetnm HEADLINE HEADSKIP NOWD SPACING=1 SPLIT='*';";
  put "COLUMN " %do i=1 %to &nvars_; "%scan(&name_,&i,'~') " %end; ";";
  %do i=1 %to &nvars_;
   %if "%scan(&label_,&i,'~')"="none" %then %let lab_=%scan(&name_,&i,'~');
   %else %let lab_=%scan(&label_,&i,'~');
```

If the variable's format is missing, use combination of data type and length to assign a format:

```
    %if "%scan(&type_,&i,'~')"="char" %then %do;
      %if "%scan(&format_,&i,'~')"="none" %then %let
form_=$%scan(&type_,&i,'~')%scan(&length_,&i,'~').;
      %else %let form_=%scan(&format_,&i,'~');
    %end;
    %else %if "%scan(&type_,&i,'~')"="num" %then %do;
      %if "%scan(&format_,&i,'~')"="none" %then %let form_=%scan(&length_,&i,'~').;
      %else %let form_=%scan(&format_,&i,'~');
    %end;
```

Arbitrarily adjust widths of very long and very short text variables:

```
    %if "%scan(&type_,&i,'~')"="char" %then %do;
      %if %eval(%scan(&length_,&i,'~') > 25)=1 %then %let len_=25;
      %else %if %eval(%scan(&length_,&i,'~') < 10)=1 %then %let len_=10;
```

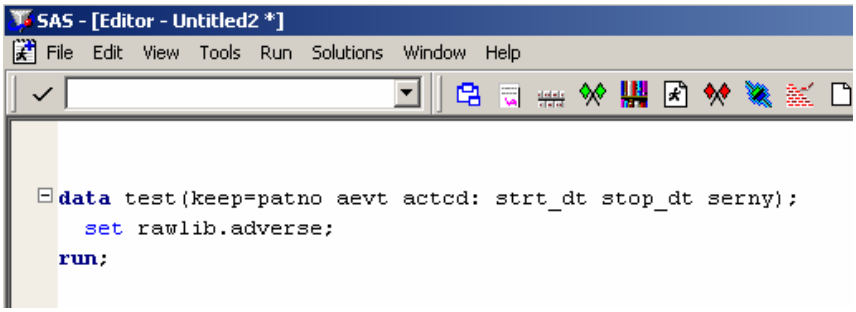
Generate "DEFINE" statements:

```
      %else %let len_=%scan(&length_,&i,'~');
        put "DEFINE %scan(&name_,&i,'~') / WIDTH=&len_ FLOW FORMAT=&form_ '" "&lab_"
"';" ;
    %end;
    %else %if "%scan(&type_,&i,'~')"="num" %then %do;
      put "DEFINE %scan(&name_,&i,'~') / WIDTH=10 FORMAT=&form_ '" "&lab_" "';" ;
    %end;
  %end;
  put "RUN;";
run;
%mend procreport;
%procreport;
```

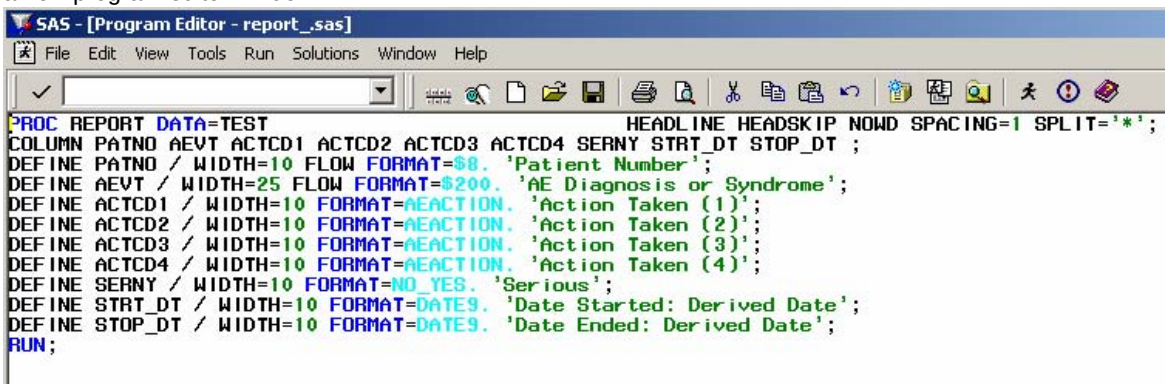
To call this macro, simply place it in "C:\WUtemp" and add an enhanced editor button with the following configuration:

```
SUBMIT "%include 'c:\wutemp\procreport.sas'; dm 'inc c:\wutemp\report_.sas';"
```

Now, when the button is clicked, the most recent dataset created in your work directory:



will be passed to the macro. PROC REPORT statements will be generated, and these statements will be displayed in a new program editor window:



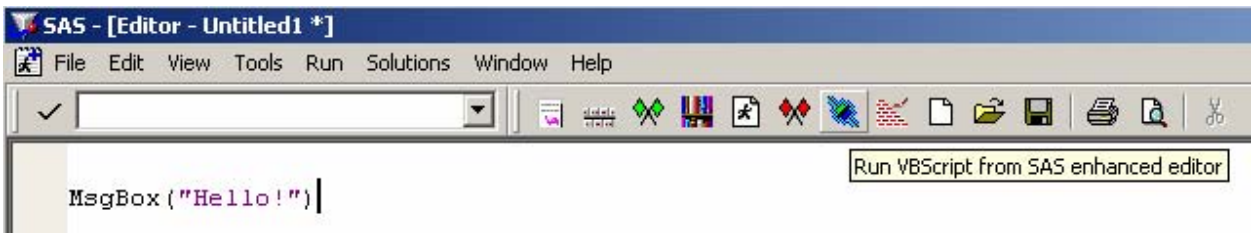
USING THE SAS EDITOR TO WRITE CODE IN "FOREIGN" LANGUAGES

SCRIPTING LANGUAGES

Learning to program in one or more scripting languages can be an entertaining way to improve your versatility and overall programming skill. For languages that can be run in interpreted environments such as the Windows Scripting Host, you don't necessarily even have to leave the SAS environment to write and test programs (this way you can take advantage of the row and column counters in the program editor, though color-coded syntax checking won't be relevant). By configuring a button with the following command:

```
SUBMIT "options noxwait xsync; dm editor 'file c:\WUTemp\test.vbs;save';x
'c:\WUTemp\test.vbs';"
```

, you can write and submit Visual Basic Scripting Edition (VBScript) code from the SAS editor:



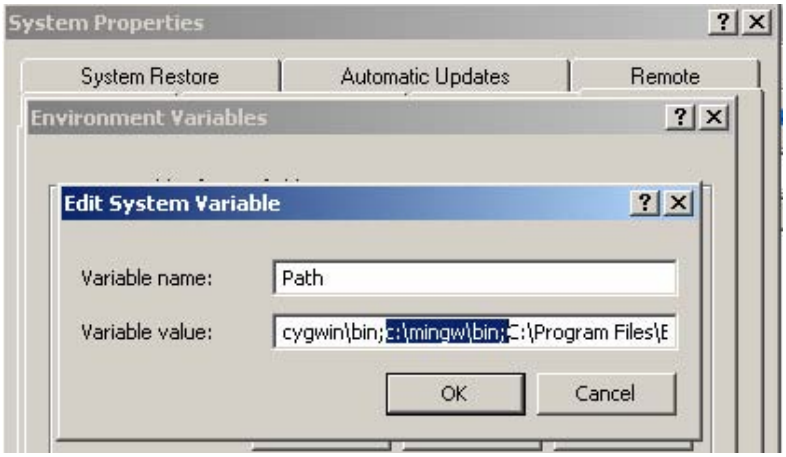
When this statement is submitted, it is saved as a text file with a ".vbs" file extension, and then run on the Windows Scripting Host (the operating system recognizes the file extension as one associated with the Windows Scripting Host, which is a language interpreter built into the Windows operating system). Script execution produces the following:

Exactly the same approach can be used for Jscript and HTML files - simply change the file extension for the source file from ".vbs" to ".js" or ".htm". Like VBScript, Jscript runs on the Windows Scripting Host, while HTML launches in your default web browser.

COMPILED LANGUAGES

Of course, with a little more hacking, we can also work with compiled languages. For example, say you want to explore the exciting world of GNU ("GNU's Not Unix", a massive collection of open-source freeware development tools), and perhaps start learning to program in C/C++, but you want to maintain at least the superficial appearance of doing 'legitimate' SAS work. Well, there are a large number of choices available, but a good place to start is with "GCC" from the "GNU Compiler Collection". GCC is a C/C++ compiler, available as part of MinGW ("Minimalist GNU"), and with very little configuration can be called from the Windows command-line interface.

First, download and install the GCC component of MinGW from http://www.mingw.org/. Next, add the path name of the directory containing the GCC.EXE file to your "PATH" system variable:

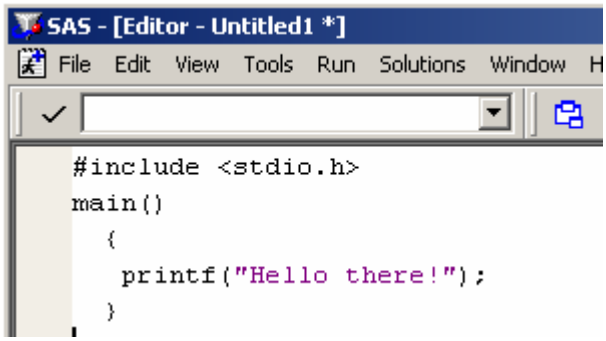


This causes Windows to include this directory when searching for binaries/executable files.

Now create a new enhanced editor button with the following command:

```
SUBMIT "options xwait xsync; dm editor 'file c:\WUTemp\test.cpp; save '; x 'gcc
c:\wutemp\test.cpp -o c:\wutemp\test.exe'; x 'cd c:\wutemp'; x 'call test.exe'; "
```

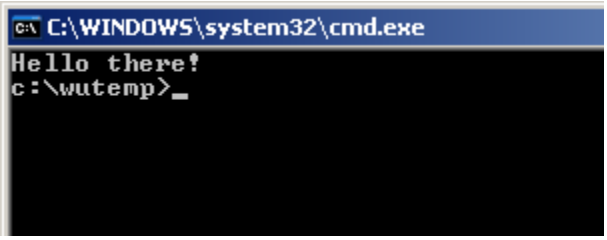
Now your're ready to begin programming in C++ in SAS:



By clicking the SAS/C++ button you've set up, the contents of the editor window are saved as a C++ source file, which is passed to the CPP compiler to create an executable:

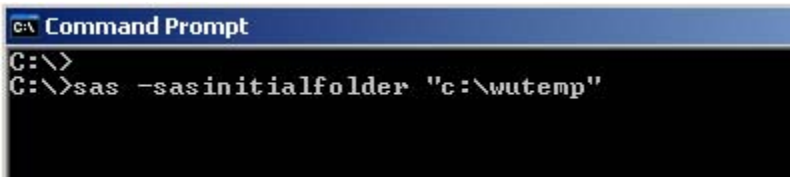
```
gcc c:\wutemp\test.cpp -o c:\wutemp\test.exe
```

An “x” command is then submitted to run the executable, displaying the following result:



SETTING THE –SASINITIALFOLDER CONFIGURATION OPTION

A common complaint leveled against many applications is that the default folder for “save as” and “open” operations is usually something useless like the ‘my documents’ folder. SAS provides a configuration option “-SASINITIALFOLDER” to change this default at start-up, but it requires either that you specify the option while launching SAS from a command line:



, or that you make the option static by placing it in the SASV8.CFG configuration file:

```
-SASINITIALFOLDER "c:\wutemp"
```

If you are working on several projects concurrently, you probably want some way of dynamically switching the –SASINITIALFOLDER setting. There are a couple of approaches you could take. The first approach exploits the ‘CurrentDirectory’ property of the Windows Scripting Host shell “WScript.Shell” in order to capture the target path name, and the configuration file is modified prior to SAS start-up. The following script is saved with a “.vbs” file extension, and then copied to the target folder and run; the “CurrentDirectory” property returns the path name of the directory the script file is sitting in.

Create shell and file system objects:

```
Dim wshell, fs, f
set wshell=Wscript.CreateObject("WScript.Shell")
set fs= CreateObject("Scripting.FileSystemObject")
```

Capture the current directory:

```
set f = fs.GetFolder(Wshell.currentdirectory)
```

Copy and modify the SAS configuration file to set –SASINITIALFOLDER:

```
 fs.CopyFile "C:\Program Files\SAS Institute\SAS\V8\SASV8.CFG",
   "C:\Program Files\SAS Institute\SAS\V8\SASV8_BAK.CFG"
 Dim intext, outtext
 Set intext=fs.OpenTextFile("C:\Program Files\SAS Institute\SAS\V8\SASV8_BAK.CFG",1)
 Set outtext=fs.CreateTextFile("C:\Program Files\SAS Institute\SAS\V8\SASV8.CFG",2)
 While Not intext.AtEndOfStream
   text_=intext.ReadLine & NewLine
   If instr(text_,"-SASINITIALFOLDER")>0 Then
   Else
    outtext.Write(text_)
    outtext.WriteLine("")
   End If
 Wend
 outtext.Write("-SASINITIALFOLDER '" & f & "'")
```

```
    intext.Close
    outtext.Close
```

Launch SAS:

```
Dim objSAS
Set objSAS=CreateObject("SAS.Application.8")
objSAS.Visible=true
```

Alternately, rather than editing the configuration file, SAS could be launched by submitting a command in the WScript Shell:

```
wshell.Run Chr(34) & "C:\PROGRA~1\SASINS~1\SAS\V8\SAS.EXE" & Chr(34) & " -
sasinitialfolder '" & f & "'"
```

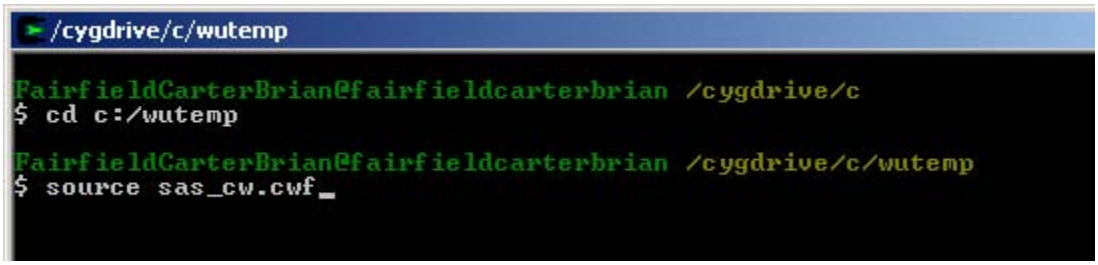
In either case, you have to navigate to the target directory and copy in the script. Let's say you don't want to do this, and that additionally you want a particular 'initialization' program (for example, that sets common library references) to run on start-up. To make things even more entertaining, we'll set things up to use the Linux-emulation Cygwin shell (http://www.cygwin.com/), though of course something similar could also be done using the Windows command-line environment.

First, download and install the Cygwin shell and basic utilities. As with MinGW, add the directories containing the Cygwin and SAS binaries to your PATH system variable.

To start off with, create a simple Cygwin source file (simple text file), containing the following commands:

```
sas -sasinitialfolder 'c:\wutemp'
```

The name of this file is arbitrary; let's save it as "c:\wutemp\sas_cw.cwf". From the Cygwin shell, change directories to 'c:\wutemp' (type "cd c:/wutemp"; note that Cygwin recognizes forward-slashes in the path name rather than back-slashes) and type 'source c:\wutemp\sas_cw.cwf':

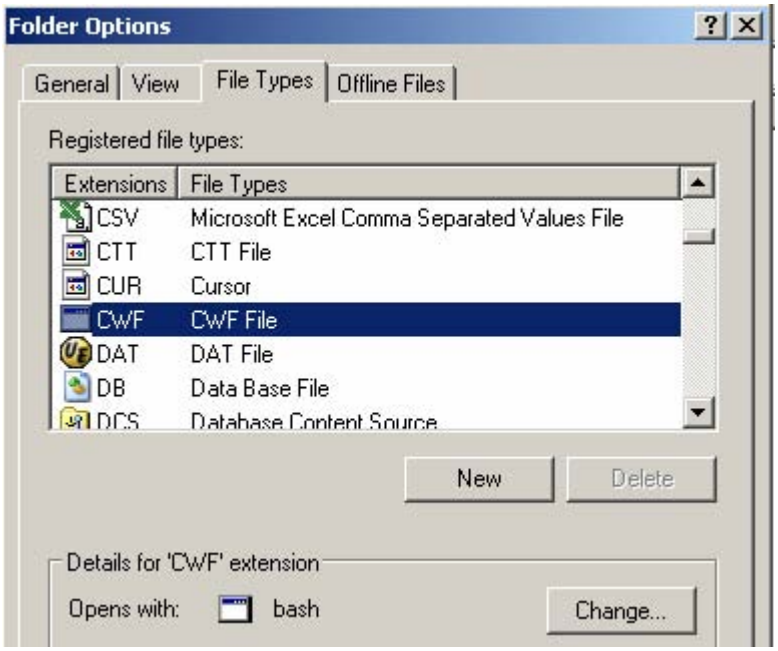


This will cause SAS to launch, setting "c:\wutemp" as the default "Open"/"Save as" folder.

Now we'll create a source file called cwSTUDY1.cwf containing the following:

```
SAS -SASINITIALFOLDER '\\<SERVER>\<STUDY1>\SAS' -AUTOEXEC
'\\<SERVER>\<STUDY1>\SAS\Prod\Define\autoexec.sas'
```

This will launch a SAS session, setting –SASINITIALFOLDER to the top-level SAS folder in "STUDY1", and automatically submitting the "autoexec.sas" program for "STUDY1". By creating similar files for each study you're working on, you can conveniently initialize your SAS environment for a specific study just by calling the appropriate source file. To take this a step further, so that you don't actually have to manually launch Cygwin and type in commands, you can 'register' the arbitrary '.cwf' file extension applied to the source files. To register a new file extension, from Windows Explorer, select Tools/Folder Options, and select the 'File Types' tab:



Click on the "New" button, and enter "CWF" as the new file extension. Then click on the "Change" button, and select the "bash.exe" binary from the Cygwin binaries folder (probably something like "C:\Cygwin\bin"). Now you can initialize your SAS environment for a specific study simply by double-clicking on the appropriate Cygwin source file.

As a final trick, so that Windows Explorer opens to the same folder that –SASINITIALFOLDER is set to at the same time that SAS is launched, add the following line to the source file(s):

```
EXPLORER '\\<Server>\\<STUDYn>\SAS'
```

Note that an "&" must also be placed at the end of the first command, so that Cygwin doesn't stop processing after launching SAS:

```
SAS -SASINITIALFOLDER '...' -AUTOEXEC '...' &
```

CONCLUSION

Hopefully this paper will have provided some inspiration to explore strange and unique ways of working with SAS, and in the process provide some useful and amusing examples. Hopefully also the reader, if not already familiar with some of the many interpreted and compiled programming environments available, will be intrigued enough to check out some of the many GNU tools available, and find additional ways of integrating them with SAS.

ACKNOWLEDGMENTS

The authors wish to thank Tim Williams for his philosophical and practical support of creativity and innovation, as well as our many friends both within and outside of Analysis and Reporting.

CONTACT INFORMATION

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

Brian Fairfield-Carter, Stephen Hunt
PRA International
600-730 View Street
Victoria, BC, Canada V8W 3Y7
Email: FairfieldCarterBrian@PRAIntl.com, HuntStephen@PRAIntl.com
Web: www.prainternational.com