

# Importing Metadata Programmatically Using the SAS® Batch Import Tool

David Moors, Whitehound Limited, UK

## ABSTRACT

Importing metadata can be a time-consuming, painstaking, and fraught task when you have multiple SAS packages to deal with.

Fortunately SAS software provides a batch facility to import (and export) metadata by using the command line, thereby saving you from many a mouse click and potential repetitive strain injury when you use the Metadata Import Wizard!

This paper aims to demystify the SAS batch import tool. It also introduces a template-driven process for programmatically building a batch file that contains the syntax needed to import metadata and then automatically execute the batch import scripts.

## THE ISSUE

Having been a SAS developer, Tester & Administrator during my SAS career one of the most time consuming and potentially frustrating tasks has to be importing metadata packages between SAS environments.

SAS provide the metadata import wizards both in SAS Management Console and Data Integration Studio, and these tools provide a fantastic job of importing the metadata packages, and providing the options to customise how and to where the metadata should be imported.

Whilst using the metadata import wizard in the Graphical User Interfaces is fine when you're importing a small number of metadata packages, when faced with a large metadata deployment with hundreds of packages, the task of importing metadata using the wizards can become a daunting and time-consuming process, and one that can be prone to error.

Some larger SAS clients have designated teams for importing metadata packages, such is the effort required to perform this task. But what if you're a smaller SAS shop and you don't have these resources at your fingertips, or you just don't fancy those repetitive clicks using the GUI's, potentially giving you RSI or a bout of boredom? Or perhaps you just like an easy life where you could just fire off a command and go and get a coffee?

### Did you miss it?

Fortunately SAS have provided a way to do exactly this via the command line using the Batch Import Tool. As the name suggests this is a command line tool.

If you import metadata via the GUI and read the Log (who does that??) then the chances are you have already come across the command used to import metadata via the command line.

If we look at the bottom of the Import log generated by the Import Wizard, we can see the command to import the SAS package (.spk) file using the Batch Import Tools.

This can be seen in Figure 1 below:

```
ImportPackage -profile "My Server" -package "S:\SGF_Paper_Files\2016\packages\SGF16_Package_StoredProcesses.spk"  
-target "/Shared Data(Folder)" -subprop
```

Figure 1

To understand and make use of the Metadata Batch Import tools a bit of background knowledge, and a delve into the SAS documentation is recommended.

## The Batch Import Tool

The metadata batch import tool is made up of various options that can be used to customise the importing of metadata, much in the same manner as the batch import wizards.

At a bare minimum the batch import must comprise of the following options:

- The **'ImportPackage'** statement
- The **'-profile'** option (or alternatively the: -host, -port, -user & -password options)
- The **'-package'** option, to specify the SAS metadata package name (and optionally the location)
- The **'-target'** option, to specify the folder location of where the metadata objects will be imported to.

The SAS documentation for the metadata batch import tool lists 15 options that can be applied to the 'ImportPackage' command. A link to the documentation for SAS 9.4 is provided below:

<http://support.sas.com/documentation/cdl/en/bisag/68240/HTML/default/viewer.htm#n0ctcrh4seaj9zn1aoss7m4ejy0.htm>

For the purposes of this paper the following statements and options for the batch import tool will be discussed, these can be seen in Table 1 below:

Name	Type	Requirement
ImportPackage	Statement	Yes
-profile	Option	Yes
-package	Option	Yes
-target	Option	Yes
-types	Option	No
-newOnly	Option	No
-subprop	Option	No
-log	Option	No
-noexecute	Option	No

Table 1

Over the next few pages we going to look at the descriptions for each of the Batch Import Tool statement and options listed above.

<b><u>ImportPackage:</u></b>	<p>This is the statement that is used to call the 'ImportPackage.exe' (or ImportPackage.sh if your running on Unix) that is part of the SAS Platform Object Framework.</p> <p>In order to execute the ImportPackage command via the command line you must navigate to the folder to where the .exe or .sh file resides. On my installation of SAS on windows the location is as follows:</p> <p><i>C:\Program Files\SAS\SASPlatformObjectFramework\9.3</i></p> <p>On Unix/Linux the path would be something similar to:</p> <p><i>/usr/local/SAS/SASPlatformObjectFramework/9.3</i></p>
Example: N/A	

<b><u>-profile:</u></b>	<p>The 'profile' option is used to specify the name of the connection profile that is used to connect to the <b>target</b> metadata server. This option is used instead of having to specify the -host, -port, -user, -password &amp; -domain options.</p> <p><b>Note:</b> the connection profile must exist on the computer where the ImportPackage command is to be executed from. If this profile does not exist the import will fail.</p> <p><b>Note2:</b> If the connection profile contains spaces then the quotation marks must be used when calling this option. For best practise I would always recommend enclosing the name of your profile in quotation marks.</p> <p>The 'profile' option is a required option and the batch import tool cannot run successfully without this.<i>/SASPlatformObjectFramework/9.3</i></p>
Example: -profile "My Server Profile"	

<b><u>-target:</u></b>	<p>The 'target; option specifies the location (relative to SAS Folders) where the package will be imported to on the target metadata server.</p> <p><b>Note:</b> To import objects from your personal folder you must submit the actual path. (e.g. /User Folders/David Moors/My Folder)</p>
<p><b>Examples:</b> -target / will import the package to the root level '/.</p> <p>-target "/Shared Data(Folder)" will import the package to the 'Shared Data' folder.</p>	

<b>-types</b>	<p>The 'types' option specifies a comma-separated list of the metadata object that are to be imported.</p> <p>When this option is selected, the import tool will only import the metadata object types that are specified. All other objects in the metadata package are ignored.</p> <p><b>Note:</b> Object types are case sensitive.</p> <p><b>Note2:</b> 'Folder' is not a valid value.</p>
<b>Example:</b> -types Job, Table,	

<b>-newOnly</b>	<p>The 'newOnly' option specifies that metadata objects are to be imported only if they do not already exist in the specified location on the target metadata server.</p> <p>If the object of the same name and type already exists, then the objects are not imported.</p> <p><b>Note:</b> If you do not specify this option the default behaviour is to overwrite existing objects that have the same location, name and type.</p>
<b>Example:</b> -newOnly	

<b>-log:</b>	<p>The 'log' option specifies the following:</p> <p>The path (or path and filename) where the log is to be written to.</p> <p>That the log is to be written to the current directory. To use this option specify a full stop (period). E.g. -log .</p> <p><b>Note:</b> the log file will contain the messages that are generated during the batch import tool's execution.</p>
<b>Examples:</b> -log "S:\logs\Import_SGF16_Package_StoredProcesses.log".	

<b>-noexecute</b>	<p>The 'noexecute' option specifies that metadata package is not to be imported to the target metadata server. Instead a list of all the objects that are included is displayed and a log file is created.</p> <p><b>Note:</b> This option is useful if you want to verify that the batch import command is constructed correctly before it is executed.</p>
<b>Example:</b> -noexecute	

<b>-subprop</b>	<p>The 'subprop' option specifies that the imported metadata objects are to be updated based upon the properties that are specified in the substitution properties file.</p> <p>A description and example of the substitute properties file is shown in the section after this text box.</p>
-----------------	--

**Note:** Before you import a SAS metadata package with the Batch Import Tool changes must be made to the substitution properties file to specify details for the target metadata environment e.g. Server Name, Port, etc.

**Note2:** If the substitute-properties-filename does not exist then the batch import tools expects the substitution property file to be named the same package-name but with the .subprop extension. An example makes this clearer:

Package Name: SGF16\_InfoMaps.spk

Subprop name: SGF16\_InfoMaps.**subprop**

**Best Practise:** When creating SAS metadata packages and using the Batch Import tool, Developers are advised to create the substitution properties file along with the metadata package (.spk).

**Note3:** The subprop file should be stored in the location from which you are importing the SAS metadata package(.spk) file from.

**Example:** –subprop SGF16\_InfoMaps\_Import.**subprop**

### The substitution property file:

The substitution property file needs to be specified when using the Batch Import tools. If you are using the metadata Import Wizard then this file is not specified. All environment mappings and association will be created via the prompts in the wizard.

By default, when a metadata package has been created using the metadata export wizard in either SAS Management Console or Data Integration Studio, a substitution properties file is contained within the .spk file.

To view the contents of a .spk file, SAS provides the **SAS Package Reader** tool. This tool was released with version 9.1.3 of the SAS Platform and the most recent update was in March 2005. However this software will still read SAS metadata packages created with more modern releases of the SAS Platform.

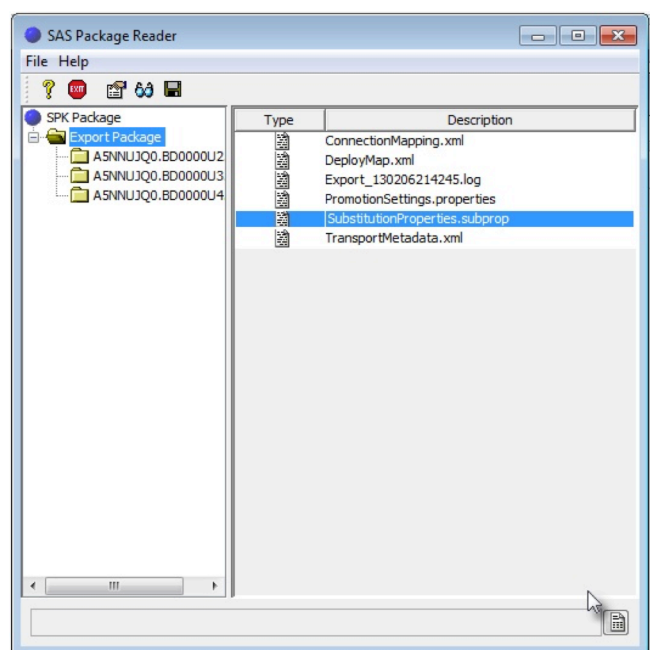
### **SAS PACKAGE READER**

The SAS Package Reader can be downloaded from the following location:

<https://support.sas.com/downloads/package.htm?pid=306>

The SAS Package Reader requires some configuration work in order to get it running, such as pointing the software to the version of the Java Runtime Environment. But once configured, launching the application is as simple as double-clicking on an .spk file on your system.

An example of the contents of a SAS metadata package opened in the SAS Package Reader can be seen in this screenshot.



In Figure 2 (below) we can see that a SubstitutionProperties.subprop file has been created by the metadata export wizard.

```
[Connections: Application Server]
ApplicationServer[1].SourceName=SASTestApp ❶
ApplicationServer[1].TargetName=SASTestApp ❷

[Connections: Table]
Table[1].SourcePath=/Shared Data/tmp/SGF16TestExtract_orig(Table)
Table[1].TargetPath=/Shared Data/tmp/SGF16TestExtract_orig(Table)

[Connections: Source Code Repository]
SourceCodeRepository[1].ApplicationServer=ApplicationServer[1]
SourceCodeRepository[1].SourceDirectory=S:\\SGF_Paper_Files\\2016\\code\\stpCode
SourceCodeRepository[1].TargetDirectory=S:\\SGF_Paper_Files\\2016\\code\\stpCode
SourceCodeRepository[1].CreateIfNeeded=false

[/Shared Data/SGF16/external/supplierDetails(ExternalFile)]
OwningFile.FilePath=S:\\Workshop\\difft\\data\\supplier.csv
OwningFile.FilePath.ApplicationServer=SASApp [/Shared Data/SGF16/library/SGF16(Library)]
DirPath=S:\\SGF_Paper_Files\\2016\\data

# The following SAS Application Server properties are provided for
# context only and may not be modified. All paths for this library
# should be valid for each of the following servers.
DirPath.ApplicationServer[1]=SASTestApp
```

**Figure 2**

If we open the SubstitutionProperties.subprop file we can see following:

Notice how the SourceName ❶ and the TargetName ❷ for the Application Server are the same when the .spk file has been created using the GUI export wizards.

When using the Batch Import tool the 'TargetName' ❷ on the SubstitutionProperties file would need to be amended to the following: ApplicationServer[1].TargetName=**SASApp**, otherwise metadata will be imported back into the *source* metadata server.

### **Batch Import Tool Example:**

Whilst some theoretical knowledge is required to get started with the Batch Import Tool, many people like to learn by example, so let's take a look at a simple example using the Batch Import tool.

Figure 3 below shows the syntax of a simple single metadata object import for a Stored Process.

```
ImportPackage ❶
-profile "My Server" ❷
-package "S:\\SGF_Paper_Files\\2016\\packages\\SGF16_Package_InformationMaps.spk" ❸
-target "/Shared Data(Folder)" ❹
-newOnly ❺
-log "S:\\SGF_Paper_Files\\2016\\packages\\logs\\Import_SGF16_Package_InformationMaps.log" ❻
-subprop ❼
```

**Figure 3**

In the example from Figure 3 above, the following actions are taking place:

- ❶ The 'ImportPackage' statement is called from the Platform Object Framework to initiate the batch import of metadata.
- ❷ The '-profile' option specifies that we want to use the "My Server" profile for the target metadata server.
- ❸ The '-package' option specifies the location and the name of the metadata package to import.
- ❹ The '-target' option specifies the target folder in which to import the metadata object(s) to.
- ❺ The '-newOnly' option specifies that only new metadata object that don't already exists are imported.
- ❻ The '-log' option specifies the location where we want the log, generated by the Batch Import Tool, to be written to.
- ❼ The '-subprop' option specifies that we want to use the SubstituteProperties file.

Note that we did not need to name the SubstituteProperties in this option as we assume the file name matches that of the metadata package to import.

For the example seen in figure 4 above, the metadata package being imported is: 'SGF16\_Package\_InformationMaps.spk'. Therefore the batch import tool assumes the SubstituteProperties to be called: SGF16\_Package\_InformationMaps.subprop

If we run the Batch Import Tool command, as see in figure 3 above, we should see the successful import of the 'SGF16\_Package\_InformationMaps.spk'. This can be seen in Figure 4 below:

Figure 4

If we now open the log file generate the Batch Import Tool this contents of the log should reflect the details generate to the command line window seen in figure 4 above.

The log file can be seen in figure 5 below:

```
INFO Current Time: 07 February 2016 12:38:31 GMT
INFO User Name: David ❶
INFO Target Metadata Server: SASApp (Port: 8561) ❷
INFO Importing objects from package
"S:\SGF_Paper_Files\2016\packages\SGF16_Package_InformationMaps.spk".
INFO Importing objects into "/Shared Data". ❸
INFO Package file version: 9300
INFO Loading substitution properties. ❹
INFO Loaded substitution properties file from package.
INFO Applying initial substitution values
INFO Importing only new objects. ❺
INFO There are no new objects in this package file to import. ❻
INFO The import process has finished successfully.
```

**Figure 5**

Analysing the log in figure 5 above, we can see the following:

The profile details were picked up the '-profile' option. ❶ & ❷

The metadata objects were being imported to the /Shared Data folder, as specified on the '-target' option. ❸

The SubstitutionProperties file was loaded using the '-subprop' option. ❹

Only new objects were to be imported as specified in the '-newOnly' option. ❺ **Note**, in this case the information map object already existed ❻ so this metadata object wasn't imported.

## TEMPLATE GENERATED APPROACH FOR METADATA OBJECT IMPORTS:

Whilst running the odd import from the command line, can be useful, the benefits from using the Batch Import Tool only really surface when you're faced with multiple metadata packages to import.

Rather than typing commands into a console shell wouldn't it be preferable to have a mechanism whereby the Batch Import Tool commands were generate programmatically, and placed into a batch file that could be called by a SAS program? This is exactly what the template-generated approach for metadata batch imports aims to achieve.

This approach consists of a number of steps and each of these will be detailed in the following pages:

- Import metadata batch import option from an Excel Template
- Programmatically generate the batch import tool commands
- Call the batch files to execute the Batch Import tool commands

## Excel Template & Import

In order to capture the plethora of options that can be passed to the Batch Import Tool, probably the best way in which to capture these changes would be in some form of template. Fortunately Microsoft Excel provides a great way in which to capture data in a template format.



Microsoft Excel is also usually available on most companies PC's, and it just so happens that Microsoft Excel provided an easy way in which to get data into SAS!

The Excel template that will be used in the batch import process can be seen in Figure 6 below:

	A	B	C	D
1	location	package	targetMetaLoc	objectTypes
2	S:\SGF_Paper_Files\2016\packages	SGF16Test1	/Shared Data	

E	F	G	H	I	J	K
newOnly	incACL	subProp	presPath	logLoc	disableX11	noExecute
Y				S:\SGF_Paper_Files\2016\packages\logs\SGF16Test1.log		Y

Figure 6

The Microsoft Excel Template seen in Figure 6, contains all the Batch Import Tool 'options' that can be used in conjunction with the 'ImportPackage' command to generate the require batch import command.

The values that are available for populating in the Excel template can be seen in Table 2 below:

Name	Format	Example
location	Free Text	S:\SGF_Paper_Files\2016\packages
package	Free Text	SGF16_Package_InformationMaps
targetMetaLoc	Free Text	/Shared Data(Folder)
objectTypes	Free Text	Job, Tables
newOnly	Y or missing	Y
incACL	Y or missing	Y
subProp	Free Text or missing	SGF16_Package_InformationMaps
presPath	Y or missing	Y
logLoc	Free Text	S:\SGF_Paper_Files\2016\packages\logs
disableX11	Y or missing	Y
noExecute	Y or missing	Y

Table 2

A situation where using a template may be beneficial is one whereby developers could complete or update the template prior to the packages being handed over to the person(s) in the organisation responsible for importing metadata. Indeed it could be suggested that completion of the template could form part of an organisations official development approach when it comes to promoting metadata.

Once the fields in the template have been populated, the next stage is to get the data into SAS.

## IMPORTING THE DATA.

A convenient way in which to import an Excel template into SAS is to use the import data wizard. After following a few simple prompts from the GUI, SAS will write the PROC IMPORT code with all the necessary options specific to the template. An example can be seen in Figure 7 below:

```
%macro importXL(xlfile); ❶
  proc import out= work.batchImportVars datafile= "&dir.\templates\&xlfile"
    dbms=excel replace;
    range="Sheet1$";
    getnames=yes;
    mixed=no;
    scantext=yes;
    usedate=yes;
    scantime=yes;

  run;

  data work.batchImportVars; ❷
    set work.batchImportVars (where=(strip(location) OR (strip(package) ^= '' ))); ❸
  run;

%mend importXL;

%importXL(batchImportParamTable_twoPackages.xlsx);
```

Figure 7

In the example in figure 7 above, a SAS Macro statement %importXL ❶ has been added to enable the use of different template names to be easily used. Here a SAS Macro parameter is used to create the macro variable &XLFILE. This macro variable will hold the Excel filename that is being passed in as a Macro parameter. In this case of the code in figure 8, it is the file 'batchImportParamTable\_twoPackages.xlsx'.

In addition to the PROC Import code in Figure 7, a DATA step has been added as housekeeping ❷. When using the Excel template, especially when values get deleted and the Excel template is reused, PROC IMPORT will sometimes import rows (observations) from Excel that contain empty records. Here the DATA step code excludes values where either the 'location' OR the 'package' variables are empty ❸. This should leave only the populated observations in the SAS data set.

Once the code has run successfully the temporary data set 'batchImportVars' is written to the WORK library. The data set will contain all the values imported from the Excel template and should look similar to the example shown in Figure 8 below:

	location	package	targetMetaLoc	logLoc
1	S:\SGF_Paper_Files\2016\packages	SGF16_Package_InformationMaps	/Shared Data(Folder)	S:\SGF_Paper_Files\2016\packages\logs
2	S:\SGF_Paper_Files\2016\packages	SGF16_Package_StoredProcesses	/Shared Data(Folder)	S:\SGF_Paper_Files\2016\packages\logs

	objectTypes	newOnly	incACL	subProp	preservePaths	disableX11	noExecute
1		Y					Y
2		Y					Y

Figure 8

Once we have the data from the Excel template in a SAS data set, the next thing that is required is to obtain the list of package names and put these values into macro variables. The values are required in macro variables (with a numerical) suffix, as the SAS DATA step code will be looping through a list of the package names in a later step.

A quick way, in which to create macro variables with numerical suffixes, using values contained in a SAS data set variable, is to use CALL SYMPUT in a DATA step. The example in Figure 9 below shows an example of this technique.

```
/*-- create macro variables containing the packages to import --*/  
data _null_;  
  set work.batchImportVars;  
  suffix = put(_n_,5.); ❶  
  call symput(cats('package',suffix), package); ❷  
run;
```

Figure 9

In the code above, a variable 'suffix' ❶ is created which will contain the value \_n\_, which is a counter for the iteration through the implicit loop of the DATA step. In the code above, the values for variable 'suffix' will be the values 1 & 2, as there are two implicit loops through the DATA step as the dataset has two observations.

Next the code uses the CALL SYMPUT statement ❷ to create macro variables with a name that is the combination of the string 'package' with the suffix variable creates in the previous statement. The value contained in dataset variable 'package' is then inserted into the macro variable &PACKAGE*n*.

The result of this DATA step is the creation of two macro variables; &PACKAGE1 and &PACKAGE2 each containing the name of package from the corresponding variable in the dataset. Figure 10 below, shows the values from the SAS Log.

```
46  options mprint;  
47  %macro loopit;  
48      %do i = 1 %to 2;  
49          %put *** package&i contains: *** &package&i ***;  
50      %end;  
51  %mend loopit;  
52  %loopit  
  
*** package1 contains: *** SGF16_Package_InformationMaps ***  
*** package2 contains: *** SGF16_Package_StoredProcesses ***
```

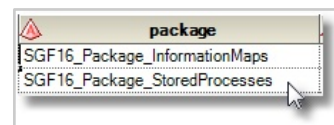


Figure 10

The main body of the SAS code is the macro %stringBuilder. As the name suggests this code is designed to build a string that will contain the ImportPackage command and all the selected options and then output the string into a batch file.

## STRING BUILDER MACRO:

The macro %stringBuilder consists of a variety of elements and these will be discussed on the following pages.

The first element in the %stringBuilder macro is the creation of a macro variable &FILESTRING*n*. This macro variable will hold the file specifications that will be called as part of the FILE statement later on in a DATA step.

Figure 11 below shows the syntax used to create the macro variable containing the FILE statement.

```
/*-- macro to create the batch files containing the metadata batch import statements --*/  
%macro stringBuilder(debug);  
  
    %do i = 1 %to &sysnobs; ❶&❷  
        %global fileString&i; ❸  
        %let fileString&i = &dir.\output\&&package&i...bat; ❹&❺  
        %put ***&fileString&i***;  
  
    %end;
```

Figure 11

In the code above a macro %DO loop ❶ is used to loop through the number of observations held in the dataset. The number of records in the data set is determined by the automatic system macro variable &SYSNOBS ❷.

Global macro variable(s) are created using the %global statement ❸ to ensure the macro variables are available throughout the whole of the SAS session.

Next, the &FILESTRING macro variable(s) are created with a %LET statement ❹. The &FILESTRING macro variable will hold the file name and also the location in which to write the batch files to.

**Note:** in the string for the %LET statement for the &FILESTRING macro variable, there is a triple full stop (period) being used ❺. This triple full stop is required due to the multiple passes to the macro processor. For more information on multiple 'amper' in macros, see to the 'Spice Girls Rule' below.

### The Spice Girls Rule: Two become One..

When dealing with SAS Macros and multiple ampersands (double macros or indirect referencing macros) the golden rule is that two ampersands resolve to become one ampersand (or 'Amper' for you professional SAS programmers out there)

Thus if we have some code that looks like the following:

```
&&package&i.
```

**Note:** in this example.. i=1

On the first pass the macro processor resolves the double 'amper' to become 1 amper, whilst the macro variable &i is resolved to be 1. The result from the first pass to the macro processor is as follows:

```
&package1
```

On the second pass to the macro processor the macro variable resolves to the required value, which would be:

```
SGF16_Package_InformationMaps
```

The result from running the code will be the creation of multiple &FILESTRING macro variables. An example of the code generated can be seen below:

```
fileString1 = S:\SGF_Paper_Files\2016\output\SGF16_Package_InformationMaps.bat  
fileString2 = S:\SGF_Paper_Files\2016\output\SGF16_Package_StoredProcesses.bat
```

The next section of code involves the creation of a NULL data set ❶ using the original imported SAS data set

'batchImportVars' as the source data<sup>②</sup>. The initial lines of the code for the DATA step can be seen in Figure 12 below:

```
data _null_; ①

    set work.batchImportVars; ②

    /*-- use a macro do loop to match the file string macro variable to the
        corresponding row in datastep --*/
    %do j = 1 %to &sysnobs; ③&④
        if _n_ = &j then do; ⑤
            /*-- call macro string to create the 'file' statement --*/
            file "&&fileString&j" LRECL=300; ⑥
        end;
    %end;
```

Figure 12

Within this code fragment above, a macro %DO loop <sup>③</sup> is being used to loop through the number of values help in the &SYSNOBS system macro variable <sup>④</sup>. The DATA step 'IF-THEN-DO' statement <sup>⑤</sup> then compares the value of the implicit iterative counter \_N\_ to macro variable &J from the macro %DO loop.

If these values match then the FILE statement is generated <sup>⑥</sup>. The macro variables for &FILESTRING (which contains the FILE statement specifications) are also called via indirect macro referencing.

The purpose of the code is to generate a FILE statement and call the corresponding &FILESTRING macro variable for each observation in the SAS data set, which should also be the number of batch files that need to be generated, based on the number of metadata packages from the Excel template.

The next section of the DATA step code is there to create all the strings that will hold the ImportPackage command and all the Batch Import Tool 'options' that were specified on the Excel template.

This section of code can be seen in Figure 13 below:

```
/*-- change directory to the location of the metadata batch script and --*/
/*-- write the string to the output file.. --*/
put 'cd /D "C:\Program Files\SAS\SASPlatformObjectFramework\9.3"'; ①

/*-- set the required import parameters --*/
length string1 $200;
string1='ImportPackage -profile "My Server" -package "' ||strip(location)|| '\ '
        ||strip(package)|| '.spk" -target "' ||strip(targetMetaLoc)|| '"'; ②

/*-- add the 'object types' parameter --*/
length string2 $100;
if strip(objectTypes) NE '' then string2= ' -types "' ||strip(objectTypes)|| '"'; ③

/*-- add 'new only' objects parameter --*/
length string3 $9;
if upcase(newOnly) = 'Y' then string3 = ' -newOnly'; ④

/*-- add option to 'include acls' parameter --*/
length string4 $12;
if upcase(strip(incACL)) = 'Y' then string4 = ' -includeACL'; ⑤
```

```

/*-- use the 'substitution properties file' parameters --*/
length string5 $100;
if strip(subProp) NE '' then string5 = ' -subprop "' ||strip(subProp)|| '"'; ❸

/*-- add the 'preserve path' parameter --*/
length string6 $15;
if upcase(strip(preservePaths)) = 'Y' then string6 = ' -preservepaths'; ❹

/*-- add the no execution parameter --*/
length string7 $11;
if upcase(strip(noExecute)) = 'Y' then string7 = ' -noexecute'; ❺

/*-- add the 'log location' path parameter --*/
length string8 $200;
if strip(logLoc) NE '' then string8 = ' -log "' ||strip(logLoc)|| '\ ' || 'Import_'
||strip(package)|| '.log"'; ❻

```

Figure 13

In the code above the first PUT statement is used to write the Windows ‘Change Directory’ command (CD) to the batch file ❶. The path being written into batch file will be that of the location of the SAS Platform Object Framework. **NOTE:** This path to the Platform Object Framework needs to be specified as the ImportPackage.exe is held in this folder, and the Batch Import Tool needs to be launched from this directory.

The first variable (string1) ❷ in the code above contains a combination of the ‘ImportPackage’ statement, the ‘-package’ option and the ‘-target’ option. These items are grouped together into a single string, as they are the bare minimum requirements that the Batch Import Tool requires in order to run successfully. All other options are non-essential but provide further customisation to the importing of the metadata packages.

The logic for variables ‘string 2,5 & 8’, or points ❸, ❹ & ❺, in the code above is to check to see if the values are not missing, and then add the appropriate Batch Import Tool ‘option’ including there value (which is help within parenthesis).

As the fields are free-form text fields in the Excel template, the value(s) may contain spaces, or comma values, etc. Thus parenthesis is required for quoting.

**Note** the quoting via parenthesis is required as part of the ‘ImportPackage’ statement, as without the values being in parenthesis the statement wouldn’t execute. A simple example of this could be the ‘-types’ option as this can contain multiple values. e.g –types “Job, Tables,StoredProcess”.

The code for variables ‘string 3,4,6 & 7’, or points ❹, ❺, ❷ & ❸ respectively, consists of simple IF-THEN processing. Here, if the value of the variable(s) contains the value ‘Y’ then the variable ‘string $n$ ’ is populated with the appropriate option for the ‘ImportPackage’. **Note:** Strings 3,4,6 & 7 are not free-form text fields in the Excel template so require only a single check for the UPCASE value of variable. The STRIP function is included to remove any leading or trailing blanks in the variable(s).

The final section of the %stringBuilder macro code can be seen in Figure 14 below:

```

/*-- combine all the variable to make the import string --*/
length stringCatX $750; ❶
stringCatX=strip(catx(' ', of string1-string8)); ❷

/*-- write the string location to the output file --*/
put stringCatX; ❸

put 'exit'; ❹

run; ❺

```

```
%mend stringBuilder; ⑥
%stringBuilder(N); ⑦
```

Figure 14

In this section of the code the length statement ① is used to create a sufficiently long enough variable to hold the 'ImportPackage' statement and all its associated options. In this case the variable is created with a length of 750 characters.

The SAS data set variable 'stringCatx' ②, is created using the CATX function to concatenate all the variables string*n* together(including the empty ones) and insert a space between each variable.

The next PUT statement ③, writes the value held in the variable 'stringCatX' to the output buffer for the FILE statement. In this example the FILE statement is writing out to an external batch file.

The final PUT statement ④, writes the DOS command 'exit' to the batch file. The 'exit' command is a shell command to terminate the current program. In this example the application is the command line shell.

The 'RUN' statement ⑤ terminates the current DATA step. Point ⑥ is the closing of the %stringBuilder macro code, and finally point ⑦ is the call to the macro processor to execute the %stringBuilder' macro code.

Once the code has been executed the SAS Log should show that the batch file(s) have been created in the location specified on the FILE statement in the DATA step.

An example of entries written to the SAS log after a successful run of the code can be see in Figure 15 below:

```
NOTE: The file "S:\SGF_Paper_Files\2016\output\SGF16_Package_InformationMaps.bat" is:
      Filename=S:\SGF_Paper_Files\2016\output\SGF16_Package_InformationMaps.bat,
      RECFM=V,LRECL=300,File Size (bytes)=0,
      Last Modified=07Feb2013:14:46:38,
      Create Time=07Feb2013:14:46:38

NOTE: The file "S:\SGF_Paper_Files\2016\output\SGF16_Package_StoredProcesses.bat" is:
      Filename=S:\SGF_Paper_Files\2016\output\SGF16_Package_StoredProcesses.bat,
      RECFM=V,LRECL=300,File Size (bytes)=0,
      Last Modified=07Feb2013:14:46:38,
      Create Time=07Feb2013:14:46:38

NOTE: 3 records were written to the file
      "S:\SGF_Paper_Files\2016\output\SGF16_Package_InformationMaps.bat".
      The minimum record length was 5.
      The maximum record length was 249.
NOTE: 3 records were written to the file
      "S:\SGF_Paper_Files\2016\output\SGF16_Package_StoredProcesses.bat".
      The minimum record length was 5.
      The maximum record length was 249.
NOTE: There were 2 observations read from the data set WORK.BATCHIMPORTVARS.
NOTE: DATA statement used (Total process time):
      real time          0.00 seconds
      cpu time           0.00 seconds
```

Figure 15

In the SAS log above we can see that two batch files have been created and these are named after the packages values that were specified in the Excel template. The path in the log states the batch files have been written to the location: S:\SGF\_Paper\_Files\2016\output.

If we look in the 'output' folder we can see the two batch files in the Windows folder, as seen in Figure 16 opposite.

If we then 'right click' on one of those files, in this case 'SGF\_Package\_InformationMaps.bat' and then choose 'edit' we can see the contents of the batch file that has been create by the SAS Program. The results can ben seen in Figure 17 below:

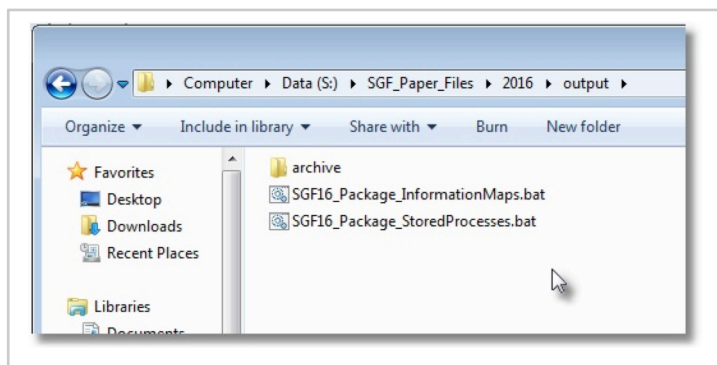


Figure 16

```
cd /D "C:\Program Files\SAS\SASPlatformObjectFramework\9.3" ❶

ImportPackage -profile "My Server" -package
"S:\SGF_Paper_Files\2016\packages\SGF16_Package_InformationMaps.spk" -target "/Shared Data(Folder)"
-noexecute -log "S:\SGF_Paper_Files\2016\packages\logs\Import_SGF16_Package_InformationMaps.log"
❷

Exit ❸
```

Figure 17

In the example above we can see that the 'Change Directory' command ❶, has been written to the batch file. The 'ImportPackage' statement and all the options specified in the Excel template have been generated ❷, and finally the 'exit' command has also been written to the batch file ❸.

## DEBUGGING:

When working with the generation of text strings from an Excel template the code generated by the SAS macro %stringBuilder can become a little messy, and the resulting batch files may be incorrectly generated.

In order to prevent this occurring the %stringBuilder macro statement has been written to contain a number of debugging steps to ensue that the code being generated looks syntactically correct before the batch files are created. Some of debugging featured built into the SAS code can be seen in Figure 18 below.

```
%macro stringBuilder(debug); ❶

/* turn on macro debugging -- only called if IN debug mode*/
%if %upcase(&debug) = Y %then %do; ❷
    options mprint mlogic symbolgen;
%end;

/* turn off macro debugging -- called if NOT in debug mode*/
%else %do;
    options nomprint nomlogic nosymbolgen;
%end;

/* assign macro variable 'debug' the value 'N' if the macro parameter is left blank
*/
%if &debug = %then &debug = N; ❸

/* loop to create the 'file' statement string -- only called if NOT in debug mode */
%if %upcase(&debug) = N %then %do;
    %do i = 1 %to &sysnobs; ❹
        %global fileString&i;
```



```

        %let fileString&i = &dir.\output\&&package&i...bat;
        %put ***&&fileString&i***;
    %end;
%end;

/* create a temporary dataset -- only called if IN debug mode */
%if %upcase(&debug) = Y %then %do; ❶
    data work.stringBuilder;
%end;

%else %do;
    data _null_; ❷
%end;

%do j = 1 %to &sysnobs;
    if _n_ = &j then do;
        %if %upcase(&debug) = N %then %do; ❸
            /*-- call macro string to create the 'file' statement --*/
            file "&&fileString&j" LRECL=300; %*NOTE: need to explain macro resolution;
        %end;
    end;
%end;

```

Figure 18

In the example above, the %stringBuilder macro statement ❶, has a macro parameter and creates the macro variable &DEBUG. The macro variable &DEBUG is designed to hold the following input values: 'Y','N',''.

The values contained in the &DEBUG macro variable form the basis of the debugging checks throughout the remainder of the program. Further examples of debugging steps are described below.

In point ❷ if the value of the macro variable &DEBUG is set to 'Y' then the macro debugging options: MPRINT, MLOGIC & SYMBOLGEN are switched on. If the value of &DEBUG is not 'Y' then macro debugging is switched off.

In point ❸ above, some validation work takes place. In this piece of code if the &DEBUG macro variable is left as a blank value (by calling the macro as follows %stringBuilder(); ) the macro code re-assigns the &DEBUG macro variable to be 'N', as it assumed no debugging is required when calling the %stringBuilder macro with an empty parameter.

Point ❹ checks to see if the value of the &DEBUG macro variable is 'N'. If the value is 'N'; then the %LET statement to create the &FILESTRING macro variable is called, otherwise the %LET statement is not executed.

In a similar manner, in point ❺, the code checks to see if the value of macro variable &DEBUG is 'Y'. If the value is equal to 'Y' then the temporary SAS data set 'stringBuilder' is created, so as to facilitate checking of the variables being created in the DATA step. This is to see if any error may have occurred. If the value of macro variable &DEBUG is set to 'N' then a NULL DATA step is created.

An example of the temporary 'stringBuilder' data set created during debugging can be seen in Figure 19 below:

location	package	targetMetaLoc	logLoc	objectTypes
1 S:\SGF_Paper_Files\2016\packages	SGF16_Package_VariousFileTypes	Shared Data(Folder)	S:\SGF_Paper_Files\2016\packages\logs	InformationMap

newOnly	incACL	subProp	preservePaths	disableX11	noExecute
1					

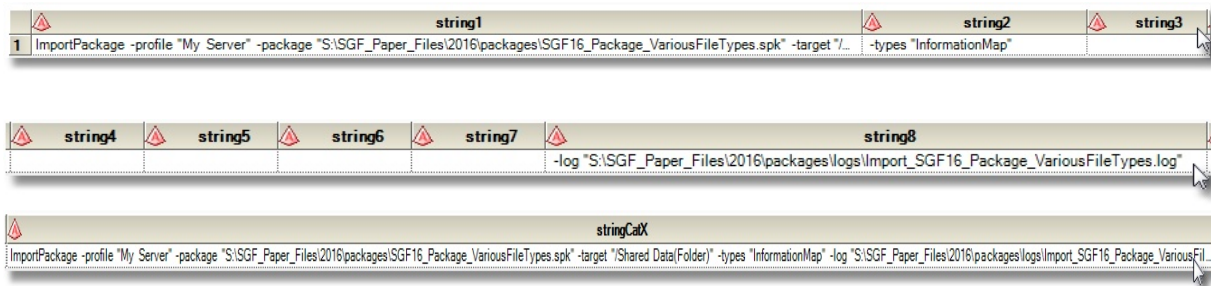


Figure 19

Finally in point ⑥ above, if the value of macro variable &DEBUG is 'N' then the 'FILE' statement is generated. If we remember the FILE statement is used to write the contents of the output buffer to a batch file. If the value of macro variable &DEBUG is not 'N' then the FILE statement is never issued and the output buffer is written to the SAS log.

## CALLING THE BATCH FILES:

The final section of the code calls the batch files that were created in the previous DATA step. This section of the code can be seen in Figure 20 below:

```
%macro callBatch; ①
  options noxwait; ②
  %do k = 1 %to &sysnobs; ③
    x "&fileString&k"; ④
  %end;
%mend callBatch;
%callBatch; ⑤
```

Figure 20

In the code above a macro statement %callBatch is create ①. Next the option 'NOXWAIT' ② is used to ensure that the EXIT command doesn't need to be typed to return to the SAS session. Next a macro %DO loop is used to loop thought the number of iterations that match the number of records from the Excel template. As discussed earlier in the paper, the number of iterations for the macro %DO loop is determined by the value held in the automatic system macro variable &SYNOBS ③.

An 'X' command ④ is used to issued to start a shell command from the SAS session, in this case it calls the &FILESTRING macro variable that was created in the code earlier on. The macro variable &FILESTRING will need to be called a number of times, depending on the values stored in the &SYNOBS macro variable. Indirect macro referencing is used in conjunction with the macro %DO loop to call the 'X' command as many times as specified in the &SYNOBS macro variable.

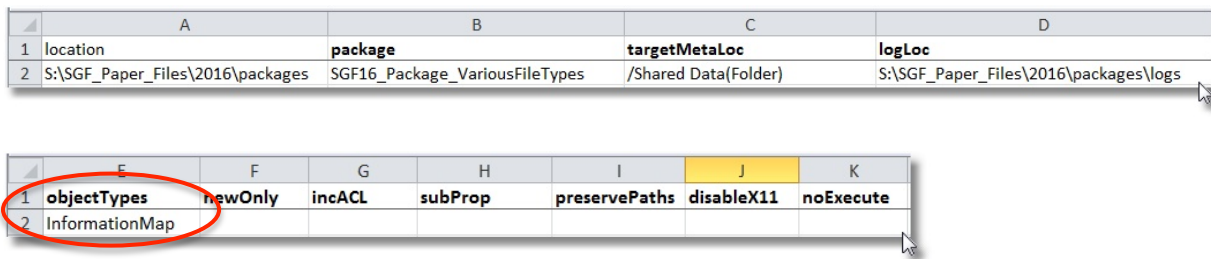
Finally a call is made to the %callBatch macro ⑤, to execute the code and run the appropriate batch files.

Once the batch files are called via SAS this will initiate the Batch Import Tool, which will then proceed to import the metadata package(s) and write the Import Log to the location specified in the batch file.

## Example 1:

As is normally the case the best way to see what is occurring in a process is by way of example. In the next session we will look at a couple of examples of the end-to-end process of importing metadata objects programmatically.

In this example the Excel Template (batchImportParamTable\_Example1.xlsx) is used to import the options specified by the user for the metadata Batch Import Tool. The options entered into the Excel template can be seen in Figure 21 below:



	A	B	C	D
1	location	package	targetMetaLoc	logLoc
2	S:\SGF_Paper_Files\2016\packages	SGF16_Package_VariousFileTypes	/Shared Data(Folder)	S:\SGF_Paper_Files\2016\packages\logs

	E	F	G	H	I	J	K
1	objectTypes	newOnly	incACL	subProp	preservePaths	disableX11	noExecute
2	InformationMap						

Figure 21

Looking at the Excel Template we can see the following details have been requested:

- Package: 'SGF16\_Package\_VariousFileTypes'
- Target locations: /Shared Data(Folder)
- Log Location: S:\SGF\_Paper\_Files\2016\packages\logs
- Option(s): -objectType "InformationMap"

Before we run the code we need to check the *target* metadata folder structure doesn't already contain the metadata object for the InformationMap we want to import.

Figure 22, opposite, shows the *target* metadata folder structure prior to the Batch Import Tool being run.

When running the SAS code, the only amendments that need to be made to the code are to the calls for the %importXL macro and also the %stringBuilder macro (to specify the debug mode).

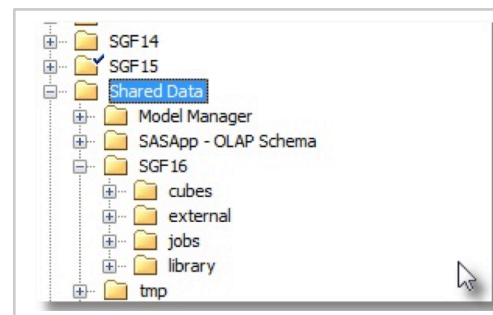


Figure 22

In this particular example we need to amend the %importXL macro to be the following:

```
%importXL(batchImportParamTable_Example1.xlsx);
```

Once the %stringBuilder macro code has run successfully the SAS log should issue a NOTE advising that the .bat file was created, along with the number of records that are written to the batch file. The code in its current incarnation will create three records in the batch file. These were discussed earlier in the paper, but as a recap the three entries are:

- 1) Change Directory (CD) command to navigate to the location of the ImportPackage.exe file in the SAS Platform Object Framework folder
- 2) The ImportPackage statement and options to be executed by the Batch Import Tool.
- 3) A command shell (exit) command to close the application.

Figure 23 below shows the 3 records being written to the batch file.

```
***S:\SGF_Paper_Files\2016\output\SGF16_Package_VariousFileTypes.bat***

NOTE: The file "S:\SGF_Paper_Files\2016\output\SGF16_Package_VariousFileTypes.bat"
is:
  Filename=S:\SGF_Paper_Files\2016\output\SGF16_Package_VariousFileTypes.bat,
  RECFM=V,LRECL=300,File Size (bytes)=0,
  Last Modified=07Feb2016:22:50:10,
  Create Time=07Feb2016:22:50:10

NOTE: 3 records were written to the file
"S:\SGF_Paper_Files\2016\output\SGF16_Package_VariousFileTypes.bat".
  The minimum record length was 4.
  The maximum record length was 255.
NOTE: There were 1 observations read from the data set WORK.BATCHIMPORTVARS.
NOTE: DATA statement used (Total process time):
  real time          0.29 seconds
  cpu time           0.00 seconds
```

Figure 23

If we look in the 'output' folder we can see the creation of the batch file that will require executing in order to run the Batch Import Tool. An example of the batch file generated by the process can be seen in Figure 24 opposite.

If we proceed to look inside the batch file we can see the batch syntax generated by the %stringBuilder macro.

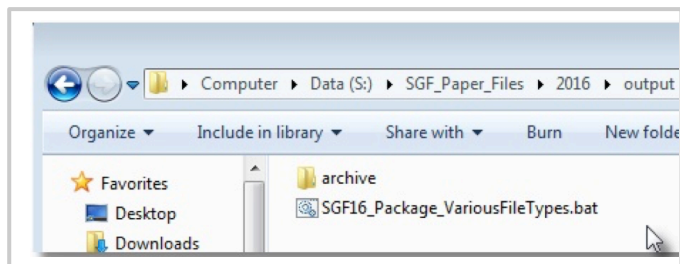


Figure 24

An example of the batch code can be seen in Figure 25 below:

```
cd /D "C:\Program Files\SAS\SASPlatformObjectFramework\9.3"

ImportPackage -profile "My Server" -package
"S:\SGF_Paper_Files\2016\packages\SGF16_Package_VariousFileTypes.spk" -target "/Shared
Data(Folder)" -types "InformationMap" -log
"S:\SGF_Paper_Files\2016\packages\logs\Import_SGF16_Package_VariousFileTypes.log"

exit
```

Figure 25

Before executing the batch file we need to check that the SAS metadata package and the substitution properties files are available, and they are located in the correct folder. For this example the 'SGF16\_Package\_VariousFileTypes' metadata package and substitution properties are available in the 'packages' folder. This can be seen in Figure 26 below:

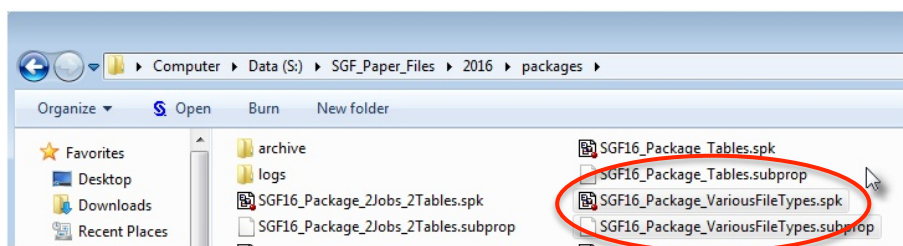


Figure 26

A call to the SAS macro %callBatch (as seen in Figure 20 earlier in the paper) will execute the batch file, which in turn will call the Batch Import Tool. Once the Batch Import Tool has run a Log will be written the location specified in the Batch file.

**Note:** as discussed earlier in the paper, if the Log destination is specified as a fullstop (period) then the metadata Import Log will be written to the directory in which the ImportPackage command is being called from.

For this particular example the log generated from running the Batch Import tool can be seen in Figure 27 below:

```
INFO Current Time: 07 February 2016 22:57:27 GMT ❶
INFO User Name: David ❷
INFO Target Metadata Server: SASMeta (Port: 8561) ❸
INFO Importing objects from package
"S:\SGF_Paper_Files\2016\packages\SGF16_Package_VariousFileTypes.spk". ❹
INFO Importing objects into "/Shared Data".
INFO Package file version: 9300
INFO Loading substitution properties.
INFO Loaded substitution properties file from package. ❺
INFO Applying initial substitution values
INFO ***** Analyzing Dependencies *****
INFO Analyzing Information map (relational) objects.
INFO ***** Starting Import Process *****
INFO Running batch process.
INFO Mapping connections using substitution properties file.
INFO Processing SAS Application Server connections.
INFO Processing Table connections.
INFO Resolving connection points.
INFO Moving the Folder "Shared Data" to "/Shared Data". ❻
INFO Validating substitution values
INFO Importing the following objects:
Create 1 Folder objects:
  /Shared Data/SGF16/infomaps
Create 1 Information map (relational) object:
  /Shared Data/SGF16/infomaps/SGF16InfoMap ❼

INFO The following connection mappings were specified:
1 SAS Application Server mapping:
  - SASTestApp --> SASApp ❽
1 Table mapping:
  - SGF16TestExtract_orig --> SGF16TestExtract_orig

INFO ***** Importing Metadata *****
INFO Metadata imported successfully.
INFO ***** Importing Properties *****
INFO Imported substitution properties for 0 objects.
INFO ***** Adjusting Metadata *****
INFO Updating Information map (relational) objects.
INFO Processing import of information map "InformationMap.Relational SGF16InfoMap
A5NNUJQ0.B100011U".
INFO ***** Importing Content *****
INFO Importing content for Information map (relational) objects.
INFO Total time to run the import process: 0.56 seconds.
INFO The import process has finished successfully. ❾
```

Figure 27

From the metadata Import Log (above) log, we can see the following actions have taken place:

- The date and time of the import has been written to the import log. ❶
- The user importing the metadata object has been identified. ❷
- The metadata server name and port number has been written to the import log. ❸
- The SAS metadata package name and location has been confirmed. ❹

- The loading of the substitution properties file has been confirmed. ⑤
- The location to import the metadata object has been confirmed as /Shared Data. ⑥
- The creation of the metadata for the information map 'SGF16InfoMap' has occurred. ⑦
- The source and target Application Server mapping is confirmed. ⑧
- The successful completion of the import process is confirmed. ⑨

If we now check the metadata folder structure, as previously seen in Figure 22, we can now see that the Information Map 'SGFInfoMap' and its containing folder 'infomaps' have been successfully imported into the target metadata folder structure.

This can be seen in Figure 28 opposite:

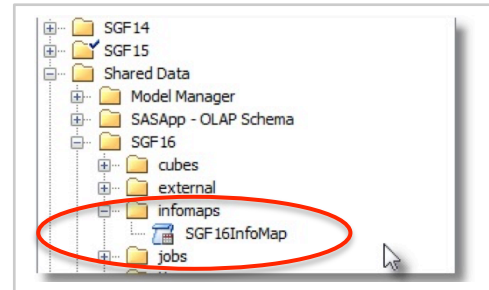


Figure 28

## EXAMPLE 2:

In the previous example, the importing of only one package was demonstrated. The code is designed so it can work with as many metadata packages as desired. As a further example we can quickly see the code work with 10 metadata packages.

The Excel template for importing ten metadata packages can be seen in Figure 29 below:

	A	B	C	D
1	location	package	targetMetaloc	logLoc
2	S:\SGF_Paper_Files\2016\packages	SGF16_Package_VariousFileTypes	/Shared Data(Folder)	S:\SGF_Paper_Files\2016\packages\logs
3	S:\SGF_Paper_Files\2016\packages	SGF16_Package2Jobs_2Tables	/Shared Data(Folder)	S:\SGF_Paper_Files\2016\packages\logs
4	S:\SGF_Paper_Files\2016\packages	SGF16_Package_2Jobs_2Tables_1Libr	/Shared Data(Folder)	S:\SGF_Paper_Files\2016\packages\logs
5	S:\SGF_Paper_Files\2016\packages	SGF16_Package_4Jobs_4Tables	/Shared Data(Folder)	S:\SGF_Paper_Files\2016\packages\logs
6	S:\SGF_Paper_Files\2016\packages	SGF16_Package_AllMetadataObjects	/Shared Data(Folder)	S:\SGF_Paper_Files\2016\packages\logs
7	S:\SGF_Paper_Files\2016\packages	SGF16_Package_ExternalFiles	/Shared Data(Folder)	S:\SGF_Paper_Files\2016\packages\logs
8	S:\SGF_Paper_Files\2016\packages	SGF16_Package_InformationMaps	/Shared Data(Folder)	S:\SGF_Paper_Files\2016\packages\logs
9	S:\SGF_Paper_Files\2016\packages	SGF16_Package_Jobs	/Shared Data(Folder)	S:\SGF_Paper_Files\2016\packages\logs
10	S:\SGF_Paper_Files\2016\packages	SGF16_Package_StoredProcesses	/Shared Data(Folder)	S:\SGF_Paper_Files\2016\packages\logs
11	S:\SGF_Paper_Files\2016\packages	SGF16_Package_Tables	/Shared Data(Folder)	S:\SGF_Paper_Files\2016\packages\logs

Figure 29

Once the SAS code has been run we can see that the SAS log is showing the number of records being written to each of the 10 batch files. The sample log can be seen in Figure 30 below:

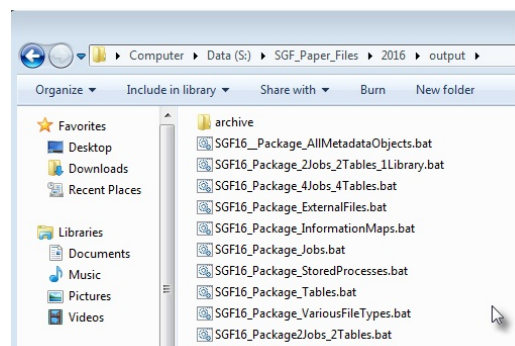
```
NOTE: 3 records were written to the file
      "S:\SGF_Paper_Files\2016\output\SGF16_Package_VariousFileTypes.bat".
      The minimum record length was 4.
      The maximum record length was 255.
NOTE: 3 records were written to the file
      "S:\SGF_Paper_Files\2016\output\SGF16_Package2Jobs_2Tables.bat".
      The minimum record length was 4.
      The maximum record length was 243.
NOTE: 3 records were written to the file
      "S:\SGF_Paper_Files\2016\output\SGF16_Package_2Jobs_2Tables_1Library.bat".
      The minimum record length was 4.
      The maximum record length was 255.
NOTE: 3 records were written to the file
      "S:\SGF_Paper_Files\2016\output\SGF16_Package_4Jobs_4Tables.bat".
      The minimum record length was 4.
      The maximum record length was 225.
NOTE: 3 records were written to the file
      "S:\SGF_Paper_Files\2016\output\SGF16_Package_AllMetadataObjects.bat".
      The minimum record length was 4.
      The maximum record length was 266.
NOTE: 3 records were written to the file
      "S:\SGF_Paper_Files\2016\output\SGF16_Package_ExternalFiles.bat".
      The minimum record length was 4.
      The maximum record length was 236.
NOTE: 3 records were written to the file
      "S:\SGF_Paper_Files\2016\output\SGF16_Package_InformationMaps.bat".
      The minimum record length was 4.
      The maximum record length was 229.
NOTE: 3 records were written to the file
      "S:\SGF_Paper_Files\2016\output\SGF16_Package_Jobs.bat".
      The minimum record length was 4.
      The maximum record length was 216.
NOTE: 3 records were written to the file
      "S:\SGF_Paper_Files\2016\output\SGF16_Package_StoredProcesses.bat".
      The minimum record length was 4.
      The maximum record length was 249.
NOTE: 3 records were written to the file
      "S:\SGF_Paper_Files\2016\output\SGF16_Package_Tables.bat".
      The minimum record length was 4.
      The maximum record length was 243.
NOTE: There were 10 observations read from the data set WORK.BATCHIMPORTVARS.
NOTE: DATA statement used (Total process time):
      real time      0.88 seconds
      cpu time       0.83 seconds
```

Figure 30



If we now check the output destination folder for the batch files we can see that 10 batch files have been created, as per the number of entries from the Excel template. The batch files can be seen in Figure 31 opposite:

Finally if we randomly check one the batch files, we can see that ImportPackage statement and its options, as specified on the Excel template, have been created in the batch file.



**Figure 31**

This can be seen in Figure 32 below:

```
cd /D "C:\Program Files\SAS\SASPlatformObjectFramework\9.3"

ImportPackage -profile "My Server" -package
"S:\SGF_Paper_Files\2016\packages\SGF16__Package_AllMetadataObjects.spk" -target "/Shared
Data(Folder)" -types "Jobs" -preservepaths -log
"S:\SGF_Paper_Files\2016\packages\logs\Import_SGF16__Package_AllMetadataObjects.log"

exit
```

**Figure 32**

If we were to then execute the windows batch files then the Batch Import Tool would successfully import the metadata into the target metadata folder structure.

## CONCLUSION

In this paper we have been introduced to the SAS metadata Batch Import Tools and have taken a detailed look at the syntax.

We have taken also taken an in-depth look at how an Excel Template, along with SAS DATA step and SAS Macro code, can be used to programmatically generate batch files that contain the syntax required to import metadata packages via the Batch Import Tools.

The calling of batch files via SAS commands has also been discussed, along with a detailed example showing the end-to-end process of using SAS code to invoke the SAS Batch Import Tool.

## REFERENCES

- [1] Carpenter, Arthur L. "Resolving and Using &&var&i Macro Variables"
- [2] Franklin, David. "Creating a DOS Batch File to run SAS Programs"
- [3] Psioda, Matthew. "Using Windows Batch Files to Sequentially Execute Sets of SAS Programs Efficiently" – SESUG2012
- [4] Rickards, Clinton S. "Creating External Files Using SAS Software"
- [5] SAS(R) 9.3 Intelligence Platform: System Administration Guide, Second Edition

## **ACKNOWLEDGMENTS**

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® Indicates USA registration.

Other brand and product names are trademarks of their respective companies.

## **CONTACT INFORMATION**

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

David Moors  
Whitehound Limited  
21 Lyons Lane  
Appleton, Cheshire, WA4 5JG  
England  
[whitehound.limited@gmail.com](mailto:whitehound.limited@gmail.com)