**PhUSE 2014**

**Paper CC05**

# A Dynamic Approach to File Organisation using DOS within SAS

Michael Tang, PPD Inc., Winchester, United Kingdom

## ABSTRACT

An ability to execute DOS commands and operate Windows applications within SAS® can provide a dynamic file organisation solution. Files external to the SAS environment can be arranged to make retrieval and review more efficient; using SAS as a data processor can automate this process and make it dynamic. This paper outlines a process to organise a Windows directory containing Tables, Listings and Figures (TLF) into a pre-defined structure such as defined in a Table of Contents or Tracker document, e.g. Batch numbers assigned to outputs depending on delivery timeframes. The process selects the required outputs, copying them into folders created using DOS commands and packages the new folder structure into a zipped file for easy delivery and review. Command options are discussed to give the user more control with process execution and steps are included to remove temporary files and folders that are created to free up disk space.

## INTRODUCTION

A clinical study can require outputs counting into the hundreds across a variety of Tables, Listings and Figures. It's typical for related outputs to be grouped and delivered in batches which break up the review process and make reviews more digestible. This paper will discuss three methods of executing DOS commands in SAS to open a communication channel with the Windows file system to allow a dynamic and automated file organisation solution.

## SETUP

Before executing DOS commands in SAS, there are a couple of system options to consider:
XWAIT/NOXWAIT – If this system option is active, the prompt below will be displayed each time a DOS command is executed.


**Figure 1**

The user must either type 'EXIT' in the DOS command window or close the command window itself to return to the SAS session. The SAS session will not continue running until one of these actions is taken. This option is set to XWAIT by default but numerous DOS commands will be executed in this example, therefore it would be beneficial to switch off this option.

XSYNC/NOXSYNC - If this system option is active, the SAS session becomes locked until the DOS command is complete. For example, suppose a DOS command is executed to open Notepad. The SAS session will not continue until Notepad is closed. For this program, we will keep this system option active.

## READING IN FILES

The example used throughout this paper will focus on a one folder location containing all outputs generated to support the Clinical Study Report. These outputs consist of a mixture of tables, listings and figures. If the study demands a high volume of outputs this location can become vastly populated, making the task of selecting a number of outputs for review time-consuming. However, a one location output area is advantageous to the method used to read a directory listing into SAS. The method is to use pipes within a filename statement. Pipes enable SAS to invoke a program external to SAS and receive the resulting messages. In this example, the pipe will execute the *dir* DOS command to obtain the directory listing.

```
FILENAME rtfdir PIPE "dir C:\Study\Output\";
```

The syntax above gives the pipe reference a name of 'rtfdir' and points to the result that the *dir* DOS command produces. If we were to run this command in a command window, the result would look like this.

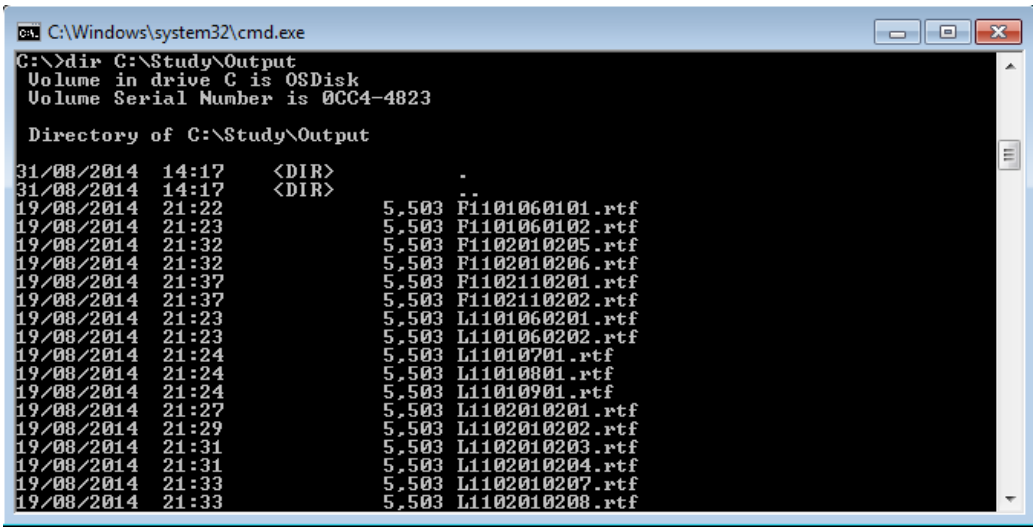


**Figure 2**

Note: the DOS command can be contained in single or double quotes. To encourage portability of the program to be used across multiple studies, it is recommended that the file location (in our example, C:\Study\Output\) is stored in a macro variable. We will continue to use double quotes in the filename statement to ensure that the file location is read as a reference, if it were to be stored in a macro variable, rather than as text.

This result still needs to be read into SAS somehow and the pipe reference is the key to accommodate this need. A data step with infile and input statements can make use of the pipe reference to print the command result into a dataset.

```
data rtffiles;
  infile rtfdir length=reclen;
  input rtfname $varying200. reclen;
run;
```

Along with the two statements in the data step, options need to be applied to ensure the result is read in correctly. The length option obtains the record length for each input line resulting from the *dir* command returning the value into a numeric variable, in our example named 'reclen'. This length variable is then used in the input statement along with the $varying. informat to set the record length in SAS. The resulting dataset looks like this:

| | rtfname |
|---|---|
| 1 | Volume in drive C is Data |
| 2 | Volume Serial Number is 7E9F-F44D |
| 3 | |
| 4 | Directory of C:\Study\Output |
| 5 | |
| 6 | 08/19/2014 04:22 PM 5,503 F1101060101.rtf |
| 7 | 08/19/2014 04:23 PM 5,503 F1101060102.rtf |
| 8 | 08/19/2014 04:32 PM 5,503 F1102010205.rtf |
| 9 | 08/19/2014 04:32 PM 5,503 F1102010206.rtf |
| 10 | 08/19/2014 04:37 PM 5,503 F1102110201.rtf |
| 11 | 08/19/2014 04:37 PM 5,503 F1102110202.rtf |
| 46 | 08/19/2014 04:34 PM 5,503 T1102100101.rtf |
| 47 | 08/19/2014 04:35 PM 5,503 T1102100102.rtf |
| 48 | 08/19/2014 04:36 PM 5,503 T1102110101.rtf |
| 49 | 08/19/2014 04:36 PM 5,503 T1102110102.rtf |
| 50 | 44 File(s) 242,132 bytes |
| 51 | 0 Dir(s) 245,105,889,280 bytes free |

**Figure 3**

Figure 3 has been slightly modified to show the bottom of the dataset containing the summary information resulting from the dir command.

#### MODIFYING THE DIR COMMAND OUTPUT

Figures 2 and 3 show the standard format of results produced when the *dir* command is executed. By default, the filename and file type extension is displayed alongside the last date and time that the file was modified and the file size for each record. In addition, header (Figure 3, rows 1-5) and summary (Figure 3, rows 50-51) information is given. We will only need the filenames to merge on tracker information to sort the outputs so we can use the **/B** option for the dir command applying a bare format to the result. The bare format strips the header and summary information and only prints the filename and file type extension for each record, providing us with a dataset that requires minimal programming manipulation to prepare for the merge with the tracker information. The output folder may also contain files with a different file type. Since we are only interested in a directory list of RTF files we can use the wildcard operator '*' in conjunction with the file extension when executing the *dir* command to only pick out files of interest.

```
FILENAME rtfdir PIPE "dir C:\Study\Output\*.rtf /B";
```

Running the filename statement with the *.rtf extension to the folder location ensures that only RTF filenames are listed in the result.

There are other *dir* command options that can sort the result by file attributes such as size, type, date and time, or display only files with particular attributes (e.g. read-only files, system files, hidden files, directories) but this is not within the scope of this example.

## OBTAINING TRACKER OR TOC INFORMATION

An output administrative tracker or a Table of Contents document will contain information required to sort the outputs. The only information needed would be the output number and the batch identifier, which can serve as a milestone indicator to prompt a subset delivery. Particularly when producing large volumes of outputs.

| | A | B | C |
|---|---|---|---|
| 1 | **Type** | **TLF Number** | **Batch ID** |
| 2 | Table | 11.1.1.1 | Batch 3 |
| 3 | Table | 11.1.2.1 | Batch 3 |
| 4 | Table | 11.1.2.2 | Batch 3 |
| 5 | Table | 11.1.3.1 | Batch 3 |
| 6 | Table | 11.1.4.1 | Batch 3 |
| 7 | Table | 11.1.5.1 | Batch 3 |
| 8 | Table | 11.1.5.2 | Batch 3 |
| 9 | Figure | 11.1.6.1.1 | Batch 3 |
| 10 | Figure | 11.1.6.1.2 | Batch 3 |

**Figure 4**

We will be using a tracker for our example shown in Figure 4, containing the output type, the TLF number and the Batch ID which will be used as the identifier to split our outputs. This tracker document is in the Microsoft Excel® format which simply requires use of the import procedure to read into SAS.

```
proc import out = work.tracker
            datafile = "C:\Study\Output\TLF Tracker.xls"
            dbms = excel replace;
     sheet = "Tracker$A1:C44";
     getnames = yes;
run;
```

The resulting dataset contains three variables that are given names according to the values in row 1 of Figure 4 (Type, TLF_Number and Batch_ID), with all other populated rows forming the records of the dataset. This dataset needs to be merged with the dataset created in the previous step containing the RTF filenames for two reasons: to identify which RTF files belong to which batch and to identify any filenames/TLF numbers that aren't present in both datasets. A common variable needs to be derived for both datasets to facilitate the merge. This example will use the RTF filename, its importance to be discussed later, so an understanding of how this is built is needed to be able translate the TLF number to match the naming convention of the RTF files.

In the tracker, the TLF number comprises of a set of numbers delimited by a period where outputs of a similar nature will commonly be incremented by the final number of the set (e.g. 11.1.2.1 and 11.1.2.2).

As a period is used prior to the file extension, the RTF file naming convention does not incorporate these such that the only one in the filename is the one that precedes the file type extension. Considering each individual number in the TLF number as an 'element' separated by periods, the TLF number can be denoted as *x.y.z...* etc. where x, y, and z are the elements.

In the file naming convention each element consists of 2 characters. If an element is a number less than 10, the element is given a prefix of '0'. So 1 becomes 01, 2 becomes 02, and so on. If the element is a number of 10 or over it remains as it is. Each element is translated this way with the periods between each element removed so a TLF number of 11.1.2.3 becomes 11010203 or a TLF number of 11.1.10.3 becomes 11011003.

The TLF number is also given a single character letter prefix based on its output type: F for a Figure, L for a Listing or Appendix, and T for a Table. Looking at the list of outputs in the tracker (Figure 4), the first output Table 11.1.1.1 becomes T11010101 and row 9, Figure 11.1.6.1.1 becomes F1101060101.

The SAS code used to translate the tracker information is below. The method counts the number of periods in the TLF number to decide how many iterations to go through to build the filename, scanning the TLF number at each iteration to decide whether the element requires the zero prefix.

```
data edit_tlfnum;
  set tracker;
  length newnum $200;
  if type in('Appendix' 'Listing') then typeinit = 'L';
  else if type = 'Figure' then typeinit = 'F';
  else if type = 'Table' then typeinit = 'T';
  count = count(tlf_number, '.') + 1;

  do i = 1 to count;
    if i = 1 then newnum = typeinit || scan(tlf_number, i, '.');
    else do;
      if input(scan(tlf_number, i, '.'),best.) < 10 then
        newnum = strip(newnum) || '0' || scan(tlf_number, i, '.');
      else newnum = strip(newnum) || scan(tlf_number, i, '.');
   end;
  end;
run;
```

By merging these two datasets together it becomes possible to identify TLF numbers in the tracker that don't exist in the RTF file location potentially indicating any outputs that haven't been produced. Likewise, it is possible to identify any RTF files that don't belong in the tracker, raising queries over whether an output has been named correctly.

```
proc sort data=edit_tlfnum; by newnum; run;
proc sort data=rtffiles; by newnum; run;

data rtf_track(keep=batch_id newnum rtfname);
  merge edit_tlfnum(in=tlfnum) rtffiles(in=rtf);
  by newnum;
  if tlfnum and not rtf then
    put "WARNING: " newnum "not in Output folder.";
  else if rtf and not tlfnum then
    put "WARNING: " newnum "not in Tracker.";
  else output;
run;
```

## PREPARING OUTPUTS FOR PACKAGING

From this section any code can be contained with a macro program. This allows for multiple packages to be created without having to import the tracker or read the directory list of RTF files again, for example if outputs are to be split by type such as Tables, Listings and Figures. It would still be good practice if outputs were to be split into batches delivered at different stages of the study as there would only be the need to change parameters in one place. Coding a macro allows the program to be run multiple times with minimal modifications to the program needed.

To prepare the outputs for packaging, an output staging area will be set up to prevent any unintentional modifications to the permanent output area. The *mkdir* DOS command can be executed to create a new directory but the method used to execute this command differs from the filename statement used earlier.
The X statement can be used within SAS to directly execute DOS commands without the need for any references or data steps. The syntax in full is simply:

```
X mkdir "C:\Study\Output\&batch."
```

This assumes the X statement being run with a macro variable called 'batch'. We will focus on packaging the Batch 3 outputs so we will give the macro variable a value of 'Batch 3'. The X statement creates a folder in the C:\Study\Output location with the name 'Batch 3'.

With the staging area created we can begin to copy the relevant outputs into it. Copying the outputs rather than moving them allows for the original to be preserved in the permanent output location; untouched in case the program doesn't execute as intended.

The merged dataset (rtf_track) comprising of the list of available outputs and the tracker information is used to identify which outputs belong to Batch 3. The tracker (Figure 4) contains a column called Batch ID which enables us to select the outputs to copy into the staging area.

The reason for retaining the RTF filenames and using these as the merge variable for the RTF directory list and tracker datasets is because these filenames will be used to compile the *copy* DOS commands. If we retained the TLF numbers as displayed in the tracker we would still have to translate them to match the RTF filenames.

Each row in the merged dataset will be manipulated to form a DOS command tailored to execute the copy for each Batch 3 output into the staging area. This represents the third method of executing a DOS command within SAS and is contained within a data step. This method takes the RTF filename in the directory list dataset (rtffiles) and compiles a *copy* DOS command as a text string into a new variable. A call system(var) statement is called within the data step to execute the command. The result is a dataset containing the DOS commands in a variable 'var' that were subsequently executed by the call system command to copy the Batch 3 outputs into the staging area. The syntax used in our example is as follows:

```
data rtf_copy;
  set rtf_track(where=(batch_id="&batch."));
    DOScmd='copy "' || left(trim("&fileout."))        || left(trim(rtfname)) ||
           '" "'    || left(trim("&fileout.&batch.")) || '"';
    call system(DOScmd);
    put rtfname "copied to the &fileout.&batch. folder";
run;
```

The code above uses the macro variables described earlier: &batch set to 'Batch 3', and &fileout set to 'C:\Study\Output\'. The DOS command is compiled in a variable called 'DOScmd' which is then executed. A message to the log is also printed to show every filename, given by the variable 'rtfname', which has been copied to the staging area. The resulting dataset that is created is merely for reference as the DOS commands will have all been executed by this point.

| Batch ID | newnum | rtfname | DOScmd |
|---|---|---|---|
| Batch 3 | F1101060101 | F1101060101.rtf | copy "U:\PhUSE\Output\F1101060101.rtf" "U:\PhUSE\Output\Batch 3" |
| Batch 3 | F1101060102 | F1101060102.rtf | copy "U:\PhUSE\Output\F1101060102.rtf" "U:\PhUSE\Output\Batch 3" |
| Batch 3 | L1101060201 | L1101060201.rtf | copy "U:\PhUSE\Output\L1101060201.rtf" "U:\PhUSE\Output\Batch 3" |

**Figure 5**

Since we had only included outputs that were common to the output folder and tracker, all Batch 3 outputs that are present in the output folder are now copied to the staging area. The log should be checked for any outputs that should be in this package that haven't been produced yet.

## CHECK FOR EXISTING PACKAGED FILES

To package the outputs, we'll be using the utility WinZip®. Packaging the outputs into a ZIP file allows compression of the output folder for ease of transfer but can also be password protected for security.

We can accommodate a check for existing ZIP files produced earlier in the study here by using the filename statement again to obtain a directory listing for the location where the ZIP files are kept. This time, the *dir* DOS command should be called without the bare format (/B) to obtain file attributes in the result. The file date and time will be incorporated into the filename to add version control to the ZIP files. It would also be useful to utilise the wildcard operator with the ZIP file extension so that we only pick out the relevant files from the directory.

Assuming the packaged outputs were given the name '&batch..zip' (in our example 'Batch 3.zip'), the code below performs a check for any outputs with that name and uses the file attributes to rename the file before printing a message to the log to inform the user that an existing ZIP file has been renamed.

```
data zipfile;
  infile zipdir length=reclen;
  input zipname $varying200. reclen;
  %*Process the following code only if a zip file with the same name exists;
  if index(zipname, "&batch..ZIP") then do;
    length filename $200;
    filename = "&batch..ZIP";
    date = put(input(scan(zipname,1,' '),mmddyy10.),date9.);
    time = put(input(scan(zipname,2,' ') || ' ' ||
               scan(zipname,3,' '),time7.),time5.);
    if length(compress(time))=4 then time='0'||compress(time);
    newfile = "&batch._" || date || '_' || compress(tranwrd(time,':','')) || '.ZIP';

  %*Rename existing zip file, appending a datetime stamp to distinguish version;
    command = 'move "' || strip("&fileout.") || strip(filename) ||
              '" "'    || strip("&fileout.") || strip(newfile)  || '"';
    call system(command);
    put "NOTE: Existing ZIP file renamed to " newfile;
  end;
  else delete;
run;
```

Code is also included to print a message to the log if no existing ZIP files were found. This is deduced if the zipfile dataset created from the code above contains no observations, as records that do not contain the filename '&batch..zip' are deleted, meaning that no ZIP files were renamed. Note the double period in '&batch..zip' for one period to allow the execution of the macro variable &batch with the second period providing the file type extension.

```
%*Determine if a previous zip file existed, if not, print the result to the log;
proc sql noprint;
  select count(*) into :zipobs
  from zipfile;
quit;

%if &zipobs.=0  %then
%put NOTE: No existing &batch..ZIP file in &fileout., new file created.;
```

## PACKAGING THE OUTPUTS

With the correct outputs copied to a staging area and any previous packaged files renamed, the folder is ready to be packaged for delivery. The X statement is used to execute the *call* DOS command to run WinZip.

```
X call "C:\Program Files\WinZip\WINZIP32.EXE" -a -r
       "&fileout.\&batch..ZIP"
       "&fileout.&batch.";
```

For this statement, three locations need to be given:
1. The location of the WinZip program on the computer.
2. The location and name of the packaged zip file to be produced.
Note that the .zip extension isn't mandatory but including this gives a clear indication that this defined location relates to the creation of the zip file.
3. The location to be zipped.

After the first set of double quotes there are a couple of parameters applied to the execution of WinZip denoted by dashes. An explanation of the two parameters given in the example and a few other useful options are as follows:
**-a** adds files to the zip file. This is one of the actions that WinZip can perform and one (and only one) must be specified to allow the utility to run. Other actions include –f for freshen, -m for move, and –u for update.
**-r** tells WinZip to include subfolders within the location to be zipped.
**-min** tells WinZip to run minimised, i.e. WinZip will only be visible in the Taskbar. If included, this parameter must be specified first, before the action.
-**hs** includes hidden and system files in the location.
**-ex**, **-en**, **-ef**, **-es**, and **-e0** are all compression methods which in sequence represent the extra, normal, fast, super-fast and no compression. If none of these parameters are specified a normal compression is used.
**-s"password"** adds a case sensitive password to the file.

## CLEANING UP

After the outputs have been packaged the staging area can be removed with the *rmdir* DOS command. It is executed with the X statement, similar to the *mkdir* command.

```
X rmdir "&fileout.&batch." /s /q;
```

A couple of options are given alongside the command:
**/s** removes a non-empty directory, without this option the command can only remove empty directories. As the aim is to remove the staging area this option must be included.
**/q** suppresses the 'Are you sure (Y/N)?' message given in DOS (shown below) when using the rmdir command to remove non-empty directories. Without this option the program cannot complete execution until this message is responded to.

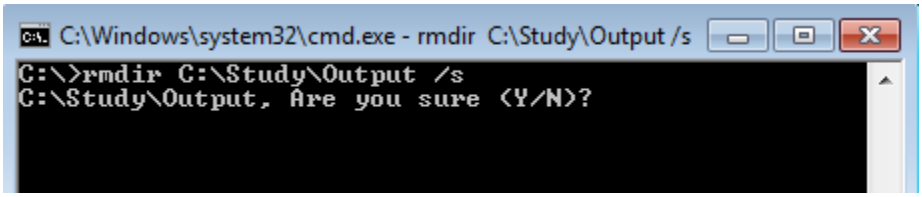


**Figure 6**

## CONCLUSION

The ability to run DOS commands within SAS opens opportunities to manipulate files and folder structures outside of the SAS environment. This paper has focused on the organisational aspect of outputs at a delivery stage but the methods can be adapted to many other uses, for example to assist top level review prior to delivery.

One example was highlighted earlier to spot TLF numbers that were either not present in the RTF directory listing or the tracker. DOS commands become a useful tool in identifying that outputs have been created but can go further by making use of file attributes, particularly the last modified date of an output. Audit checks can be introduced to ensure that the tracker is kept up-to-date with completion dates outputs by checking dates manually entered by programmers in the tracker against last modified dates of RTF files.

## REFERENCES

Na Li (2005), "Applications for Running DOS Commands within SAS", PharmaSUG 2005, Paper PO13.

SAS Institute Inc. (2010), "Running Windows or MS-DOS Commands from within SAS", SAS 9.2 Companion for Windows, Second Edition.

## CONTACT INFORMATION

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

Michael Tang
PPD Inc.
Franklin House, Kings Worthy
Winchester, Hampshire
SO23 7TW, United Kingdom
Email: michael.tang@ppdi.com