

Post-Processing .LST files to get what you want

Edward Foster, Oxford Pharmaceutical Sciences, UK

ABSTRACT

SAS has a range of procedures you can use to create table and listing output. These range from the simple PROC PRINT to more advanced reporting procedures such as PROC REPORT and PROC TABULATE. If we want the highest level of control we can even produce output using a DATA _NULL_ step. Even with this array of reporting options sometimes you cannot always produce what you want using these standard methods. This paper will outline a number of methods using SAS and MS Word to get the results you require.

INTRODUCTION

Since the introduction of ODS in version 8 a large part of the focus in report production has fallen on how this SAS technology can be used to produce clinical trial report output. However a large part of the output created by pharmaceutical companies and CROs still takes the form of ASCII files created from SAS using PROC PRINTTO statements. These files are typically denoted with the standard SAS .LST file extension; though of course this can be any file extension (.TXT is another popular option).

The SAS output procedures such as PROC REPORT or PROC TABULATE are used extensively to produce SAS output because most of the time the required results can be obtained by harnessing these procedures with the correct reporting options. However it is not always easy to produce exactly what you want through these procedures. Sometimes even simple requirements require a bit of effort on the part of the programmer. Sometimes all we need is a page number. Sometimes we need to format the text differently. Whatever it is we need there are a few simple techniques that we can employ to really get what we want.

The first technique is to post-process the output file from within SAS. This is a good option for a couple of reasons. Firstly the post-processing process can be part of the program that creates the output. This is an advantage because the process can firstly be audited through the SAS log. Secondly creation and formatting of the output remains with the SAS programmer which means that the entire output production process is in one package (namely the SAS program). The second technique is post-processing in MS Word. Importing SAS output into MS Word is a common practice in the complication of study output into a distributable package. Even if MS Word is not the final destination of the SAS output (conversion to PDF is common) it is often used as a method to firstly format (bold text etc) the output into the required state. The reason for this is because MS Word can be automated using macro code written in VBA (Visual Basic for Applications). With VBA it becomes possible to format the report in ways that are not possible with an ASCII file. For example, an ASCII file has no notion of bolded text or any other text formatting (like fonts etc). From MS Word it is possible to apply all these to the output and in an automatic way.

A key component of post processing output is the use of 'tags'. In many cases implementing post processing techniques is not possible without these.

USING 'TAGS'

Tags are best described as place-holders or markers that in this context are used to show the post-processing program what needs to be formatted and how. The use of tags is widespread across the programming world. Perhaps the most widely used set of tags in the world is the HTML markup language. By using HTML tags it is possible to format the content of web pages with the browser. For example to make a piece of text bold in a browser window we can 'mark it up' using the and tags at either end of the text in question. When the browser renders this text it knows to render it as bold but it always removes the tags before displaying the text. In this way the formatting information provided by the tags are invisible to the user. This is the main difference between these tags and many of the tags that we will need to implement to post-process our output because our tags will not be rendered invisible in the file and so in most cases will need to be removed.

TECHNIQUE 1 – POST-PROCESSING IN SAS

One of the most common items that we need on a report is a page number, usually in the 'Page x of y' format. This is not easy to do using the standard report procedures though it can be easier to do with ODS style output using custom style templates. For listing output an option is to implement a DATA _NULL_ step to exactly position all the items on your page. This is the most advanced SAS reporting option and gives you the most control but can be time consuming to implement especially if the

report format is complex. Most of the time a better option is to combine the standard reporting procedures with a simpler DATA _NULL_ step. The standard reporting procedure will produce the main portion of the report getting us 90-95% of the way there, with the DATA _NULL_ used to post-process the file to add, delete or format items after the standard reporting procedure has finished.

This methodology is much easier and faster to implement than a full DATA _NULL_ report because the formatting of the main sections of the output, such as the table columns and headers, is taken care of by the SAS reporting procedure. The SAS reporting procedure is therefore responsible for quickly routing the output to the output file. Once the procedure has completed this output file can then be imported into back into a SAS dataset using an INFILE statement. Once the file has been re-imported into SAS the lines of text that make up the output can be edited. Items such as page numbers can then be added to the report. The edited dataset can then be exported back to the output file using FILE statements.

READING OUTPUT BACK INTO SAS

The key to post-processing in SAS is accurately importing the output file you have created back into SAS. There are a number of different ways to do this but below is an example of one method:

```
data import;
  infile "D:\PostProcess\TDemog.lst" length=len;
  length text $200;

  input text $varying. len;
run;
```

The \$VARYING format is useful for importing this type of file as it reads each line using the actual length of each line (supplied by the LEN variable) as a guide for the format. This means that no data should be lost by truncation in the import process.

The resulting IMPORT dataset has one line per line of text in the LST file. Once the data has been imported into SAS we can manipulate the formatting of the file through using SAS character functions like SUBSTR or SCAN.

SAS EXAMPLE

As previously stated the addition of a page number is a very common requirement in the creation of SAS output. Another requirement could be to add a '(continued)' flag to both the bottom of each page and the end of each report title. The '(continued)' flag should only appear from page 2 onwards. The output can be found in Appendix 1.

Firstly we need to know how many page we have in our output file. Typically either the protocol number or table number of the output is displayed on the first line of an output. In our case it is the protocol number. We can use this to determine when we have entered a new page. We also need to know the maximum width of all the lines in order that we can accurately place the page number of the far right of the report. We could therefore modify the code used to import the LST file to determine the both the number of pages and the width of each line in the same step.

```
data import;
  infile "D:\PostProcess\TDemog.lst" length=len;
  length text $200 _page _len 8;
  retain _page 0;

  input text $varying. len;

  _len=len;
  if index(text, "Protocol: xxx123")>0 then _page+1;
run;

proc sql noprint;
  select trim(left(put(max(_len),8))), trim(left(put(max(_page),8)))
  into : maxlen, : maxpage
  from import
  ;
quit;
```

With both of these pieces of information we can now add the page number where we want.

```
data addpages;
  set import;
  by _page;
```

```

** WORK OUT THE POSITIONING OF THE PAGE NUMBER BY WORKING **;
** OUT THE MAXIMUM WIDTH REQUIRED **;
_pagepos = &maxlen-length("Page &maxpage of &maxpage");
_pagetxt="Page "||trim(left(put(_page,8.)))||" of &maxpage";

** ADD THE PAGE NUMBER TO THE OUTPUT **;
if first._page then substr(text,_pagepos,length(_pagetxt))=_pagetxt;

** REPLACE 'LAST LINE' WITH '(continued)' **;
_llinepos= index(text, 'LAST LINE');
if _llinepos>0 AND _page>1 then substr(text, _llinepos, 11)='(continued)';
else if _linepos>0 AND _page=1 then substr(text, _linepos, 9)='(continued)';

** ADD CONTINUED TO END OF TITLE **;
if _page>1 AND index(text,"Table 2.1")>0 then substr(text,_len+2,11)= '(continued)';

run;

```

The '(continued)' flags are added in a slightly different way. For the '(continued)' flag at the bottom of the page a tag was used to denote where to place the flag. This flag is the text 'LAST LINE' which is placed by the reporting program as the very last FOOTNOTE in the report. When the post-processing program comes across this tag it knows it needs to replace the tag with '(continued)' on each page except the first (in which case it just deletes the tag). In the case of the title we used the table number as the tag to place the flag.

The final step is to output the dataset back into a LST file. The code below demonstrated how this can be done. The code below actually replaces the original file. This is common practice (but not essential) because the original file format does not need to be kept.

```

data _null_;
  set addpages;
  file "D:\PostProcess\TDemog.lst";

  _newlen=length(text);
  put text $varying. _newlen;

run;

```

This code could be easily converted into a generic macro program that could be used to perform the same task on a number of tables. For example the tag values and file names could be supplied by macro variables instead.

TECHNIQUE 2 – POST-PROCESSING IN WORD

There are occasions where it is not possible to post-process our LST in the way we want with SAS itself. An example of this is where we need to format our LST file with a particular font or bold certain text etc. The reason for this is because the ASCII file that is produced by SAS does not have any concept of any sort of text formatting. This is where MS Word comes in. MS Word can be automated with macros created using VBA. With these macros it becomes possible to further transform the tables we create.

IMPORT INTO WORD

As with post processing in SAS the first stage of post-processing in word is to import the LST file. This VBA code will import our file into Word (this file is in Appendix 2) set the font and page orientation:

```

Selection.InsertFile ("D:\PostProcess\TDemog2.lst")

With ActiveDocument.PageSetup
  .LineNumbering.Active = False
  .Orientation = wdOrientLandscape
  .TopMargin = CentimetersToPoints(2.5)
  .BottomMargin = CentimetersToPoints(2.5)
  .LeftMargin = CentimetersToPoints(1.1)
  .RightMargin = CentimetersToPoints(1.1)
  .Gutter = CentimetersToPoints(0)
  .HeaderDistance = CentimetersToPoints(1.25)
  .FooterDistance = CentimetersToPoints(1.25)

```

```
.PageWidth = CentimetersToPoints(27.94)
.PageHeight = CentimetersToPoints(21.59)
.FirstPageTray = wdPrinterDefaultBin
.OtherPagesTray = wdPrinterDefaultBin
.OddAndEvenPagesHeaderFooter = False
.DifferentFirstPageHeaderFooter = False
.VerticalAlignment = wdAlignVerticalTop
.SuppressEndnotes = False
.MirrorMargins = False
.TwoPagesOnOne = False
.GutterPos = wdGutterPosLeft
End With
Selection.WholeStory
Selection.Font.Name = "Courier New"
Selection.Font.Size = 8
```

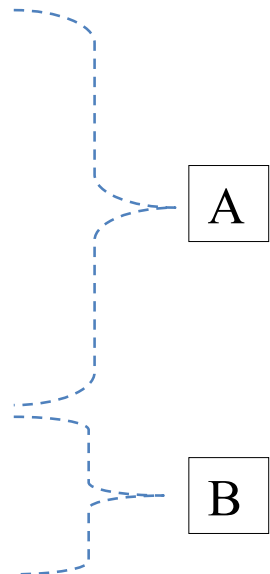
Once the file has been imported it is now possible to format how we want. In our example we want to perform the listed actions when we find the following tags

- <BOLDTHISWORD> - Bold the word to the right of the tag
- <BOLDTHISLINE> - Bold the all the text on the same line to the right of the tag
- <ITALICLINE> - Convert all the text on the same line to italic

To do this we need to hook up the search facility within Word so we can loop through the document searching for each occurrence of the tags. The following code searches for '<BOLDTHISWORD>'. Section 'A' returns the cursor to the top of the document and then sets the parameters for the search. Section B executes the search. The code within the DO..WHILE loop is executed each time the search returns a successful find of the text. When a tag is found it is first deleted then the next word on the line is selected and formatted as bold.

```
' SEARCH FOR <BOLDTHISWORD>
```

```
Selection.HomeKey Unit:=wdStory
Selection.Find.ClearFormatting
Selection.Find.Style = ActiveDocument.Styles("Plain Text")
With Selection.Find
    .Text = "<BOLDTHISWORD>"
    .Replacement.Text = ""
    .Forward = True
    .Wrap = wdFindContinue
    .Format = False
    .MatchCase = False
    .MatchWholeWord = False
    .MatchWildcards = False
    .MatchSoundsLike = False
    .MatchAllWordForms = False
End With
Do While Selection.Find.Execute = True
    Selection.Delete
    Selection.MoveRight Unit:=wdWord, Count:=1, Extend:=wdExtend
    Selection.Font.Bold = True
    Selection.MoveLeft Unit:=wdCharacter, Count:=1
Loop
```



This technique can be used to search for any number of tags within the document and execute the formatting code. The Whole macro is found in Appendix 3.

CONCLUSION

As you can see it can be easy to format LST files the way you require. If it is not possible to get what you want from the standard reporting procedures it is possible to implement a post-processing strategy that will fill the gaps. The two techniques outlined in this paper detail how this can be put into place with SAS or MS Word.

We can create very complicated generic formatting programs which can accurately position all sorts of items by the use of a system of consistently used 'tags'. The tags are introduced into the raw output file either prior to reporting within the reporting dataset or are introduced within the reporting procedure itself. By using a set of different tags each with different meanings we

are effectively creating markers for the post processing programs to search for and interpret.

Developing a library of tags with different but explicit meanings allows a post processing program to create a range of complicated changes to the output file. A consistent application of tags from a tag library across SAS programs means that post processing transformations can be encapsulated within generic SAS or VBA macros.

CONTACT INFORMATION

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

Edward Foster
Oxford Pharmaceutical Sciences
Lancaster House
Kingston Business Park
Kingston Bagpuize
Oxford / OX13 5FE
Work Phone: +44 (0) 1462 477948
Email: edward.foster@ops-web.com
Web: www.ops-web.com

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.

PhUSE 2006

APPENDIX 1

Protocol: xxx123

Table 2.1 Demography Characteristics

Overall	
Number(%) of Subjects	xxx
Age (years):	
<=1 Year	xx (xx.x%)
>1 Year - 6 Years	xx (xx.x%)
>6 Years - 8 Years	xx (xx.x%)
>8 Years - 12 Years	xx (xx.x%)
>12 Years - 17 Years	xx (xx.x%)
>17 Years	xx (xx.x%)
Mean	x.x
SD	x.x
Range	xx-xx
Gender:	
Male	xx (xx.x%)
Female	xx (xx.x%)
Total	xx (xx.x%)

LAST LINE

PhUSE 2006

Protocol: xxx123

Table 2.1 Demography Characteristics

	Overall
Number(%) of Subjects	xxx
Race:	
White	xx (xx.x%)
Black	xx (xx.x%)
Asian	xx (xx.x%)
Other	xx (xx.x%)
Weight (kg):	
Low Weight Group (>x-xxkg)	
Mean	xx.x
SD	x.x
Range	x.x-xx.x
N	xx
Medium Weight Group (>xx-xxkg)	
Mean	xx.x
SD	x.x
Range	x.x-xx.x
N	xx
High Weight Group (>xxkg)	
Mean	xx.x
SD	x.x
Range	x.x-xx.x
N	xx

LAST LINE

PhUSE 2006

Protocol: xxx123

Table 2.1 Demography Characteristics

Overall	
Number(%) of Subjects	xxx
Height (cm):	
Mean	xx.x
SD	x.x
Range	x.x-xx.x
N	xx

LAST LINE

APPENDIX 2

<BOLDTHISWORD>Protocol: xxx123
of 3

Page 1

<BOLDTHISLINE>Table 2.1 Demography Characteristics

	Overall
Number(%) of Subjects	xxx
<BOLDTHISLINE>Age (years):	
<=1 Year	xx (xx.x%)
>1 Year - 6 Years	xx (xx.x%)
>6 Years - 8 Years	xx (xx.x%)
>8 Years - 12 Years	xx (xx.x%)
>12 Years - 17 Years	xx (xx.x%)
>17 Years	xx (xx.x%)
Mean	x.x
SD	x.x
Range	xx-xx
<BOLDTHISLINE>Gender:	
Male	xx (xx.x%)
Female	xx (xx.x%)
Total	xx (xx.x%)

<BOLDTHISWORD>Protocol: xxx123
3

<BOLDTHISLINE>Table 2.1 Demography Characteristics (continued)

Overall	
Number (%) of Subjects	xxx
<BOLDTHISLINE>Race:	
White	xx (xx.x%)
Black	xx (xx.x%)
Asian	xx (xx.x%)
Other	xx (xx.x%)
<BOLDTHISLINE>Weight (kg):	
<ITALICLINE>Low Weight Group (>x-xxkg)	
Mean	xx.x
SD	x.x
Range	x.x-xx.x
N	xx
<ITALICLINE>Medium Weight Group (>xx-xxkg)	
Mean	xx.x
SD	x.x
Range	x.x-xx.x
N	xx
<ITALICLINE>High Weight Group (>xxkg)	
Mean	xx.x
SD	x.x
Range	x.x-xx.x
N	xx

(continued)

<BOLDTHISWORD>Protocol: xxx123
3

<BOLDTHISLINE>Table 2.1 Demography Characteristics (continued)

Overall	
Number (%) of Subjects	128

<BOLDTHISLINE>Height (cm):

Mean	xx.x
SD	x.x
Range	x.x-xx.x
N	xx

(continued)

APPENDIX 3

Sub Formatting()
,

' Macro1 Macro
,

,

Selection.InsertFile ("C:\Documents and Settings\Edward\My Documents\PhUSE Presentations\PhUSE 2006 (Dublin)\3.
Post Processing .LST Files\Code\Tdemog2.lst")

With ActiveDocument.PageSetup

.LineNumbering.Active = False
.Orientation = wdOrientLandscape
.TopMargin = CentimetersToPoints(2.5)
.BottomMargin = CentimetersToPoints(2.5)
.LeftMargin = CentimetersToPoints(1.1)
.RightMargin = CentimetersToPoints(1.1)
.Gutter = CentimetersToPoints(0)
.HeaderDistance = CentimetersToPoints(1.25)
.FooterDistance = CentimetersToPoints(1.25)
.PageWidth = CentimetersToPoints(27.94)
.PageHeight = CentimetersToPoints(21.59)
.FirstPageTray = wdPrinterDefaultBin
.OtherPagesTray = wdPrinterDefaultBin
.OddAndEvenPagesHeaderFooter = False
.DifferentFirstPageHeaderFooter = False
.VerticalAlignment = wdAlignVerticalTop
.SuppressEndnotes = False
.MirrorMargins = False
.TwoPagesOnOne = False
.GutterPos = wdGutterPosLeft

End With

Selection.WholeStory

Selection.Font.Name = "Courier New"

Selection.Font.Size = 8

' SEARCH FOR <BOLDTHISWORD>

Selection.HomeKey Unit:=wdStory

Selection.Find.ClearFormatting

Selection.Find.Style = ActiveDocument.Styles("Plain Text")

With Selection.Find

.Text = "<BOLDTHISWORD>"
.Replacement.Text = ""
.Forward = True
.Wrap = wdFindContinue
.Format = False
.MatchCase = False
.MatchWholeWord = False
.MatchWildcards = False
.MatchSoundsLike = False
.MatchAllWordForms = False

End With

Do While Selection.Find.Execute = True

Selection.Delete

Selection.MoveRight Unit:=wdWord, Count:=1, Extend:=wdExtend

Selection.Font.Bold = True

Selection.MoveLeft Unit:=wdCharacter, Count:=1

```

Loop

' SEARCH FOR <BOLDTHISLINE>

Selection.HomeKey Unit:=wdStory
Selection.Find.ClearFormatting
Selection.Find.Style = ActiveDocument.Styles("Plain Text")
With Selection.Find
    .Text = "<BOLDTHISLINE>"
    .Replacement.Text = ""
    .Forward = True
    .Wrap = wdFindContinue
    .Format = False
    .MatchCase = False
    .MatchWholeWord = False
    .MatchWildcards = False
    .MatchSoundsLike = False
    .MatchAllWordForms = False
End With
Do While Selection.Find.Execute = True
    Selection.Delete
    Selection.EndKey Unit:=wdLine, Extend:=wdExtend
    Selection.Font.Bold = True
    Selection.MoveLeft Unit:=wdCharacter, Count:=1
Loop

' SEARCH FOR <ITALICLINE>

Selection.HomeKey Unit:=wdStory
Selection.Find.ClearFormatting
Selection.Find.Style = ActiveDocument.Styles("Plain Text")
With Selection.Find
    .Text = "<ITALICLINE>"
    .Replacement.Text = ""
    .Forward = True
    .Wrap = wdFindContinue
    .Format = False
    .MatchCase = False
    .MatchWholeWord = False
    .MatchWildcards = False
    .MatchSoundsLike = False
    .MatchAllWordForms = False
End With
Do While Selection.Find.Execute = True
    Selection.Delete
    Selection.EndKey Unit:=wdLine, Extend:=wdExtend
    Selection.Font.Italic = True
    Selection.MoveLeft Unit:=wdCharacter, Count:=1
Loop

End Sub

```