Tedious Template Made Easy

Cindy Stroupe
Inveresk Research
Cary, NC

Abstract

In the age of electronic submissions to the FDA, standardizing the look of the output across multiple ODS destinations has become more of a challenge. SAS® has given us a method of customizing our electronic output through the TEMPLATE procedure. This procedure allows the customization of all aspects of the report from column headers to the font size of the footnotes. However, when producing many files for the same document, creating a template for each file can be a tedious process. The creation of a macro that calls in header and footnote information, sets a standard style across all columns, and outputs a similar looking file regardless of the output destination is an easy and innovative way to avoid the tedium.

In this paper, an overview of the TEMPLATE procedure, with style options for customizing your output will be presented. Additionally, a macro for using your template for all files whether you have 1 column or 10 columns will be discussed. This template macro can also be used across ODS output destinations so that your PDF files can look similar to your RTF files.

Introduction

Overview of the TEMPLATE procedure

Proc TEMPLATE can be used to customize your report in two different areas: style and table. Style customization involves changing the way the standard SAS® styles are displayed (i.e., background color, grid lines, font color, etc.). In table customization you specify how you would like your data displayed (i.e., column justification, headers, footnotes, etc.).
Here is the basic structure of the TEMPLATE procedure for table customization:

```
proc template;
 define table means.inveresk;
   <options>;

  mvar <var>;
   nmvar <var>;
  column <list of columns>;

  header <list of table headers>;
  footer <list of table footers>;

 define table_header_1;
    text “Title 1”;
    just=left;
    width=80;
    start=col1;
    end=col4;
  end;

  define table_footer_1;
    text “Footnote 1”;
    width=100;
  end;

   define col1;
     width= 70;
     header= ”Column 1”;
   end;
end;
run;
```

The first step in table customization is creating the table definition. In the example above means.inveresk is the name to use when writing out the table. The first part of the table definition ‘means’ is the template store in which to save the definition. If the template store does not exist, SAS® will create it.

The next thing to do is define your options. Options in this section control page breaks, splitting of information across pages, column spacing, header and footer spacing, how to distribute extra space and justification of formatted variables. How options affect the look of the output depends on the destination that you are using. For example the underline and overline options only work in the printer destination and not in RTF.

MVAR and NMVAR are used to define macro variables that you would like to use with your template. The difference between MVAR and NMVAR is how ODS stores the information for reference when the variable is resolved. MVARs are stored as strings, while NMVARs are stored as numeric. When you use the variables listed within this statement in your define statements, you do not need to use an ampersand.

Within the TEMPLATE procedure, you must list the columns that you will be defining. The columns will be displayed according to the order that is specified within this statement. An option called generic allows multiple columns within your report to use the same definition. If all of the columns in your report can be the same size and the label can be used as the header than even if you have 10 columns, only one define column statement is required. If you have 9 columns that can be the same and 1 column that needs a more specific definition than your column statement can have 2 columns.

You must also list each header and footer that the table is to contain. For each table header, and table footer you have, you will need to create a define statement. Some options that can be used with headers and footers are justification, width, and which columns for the headers and footers to span across.

Creating a new style is not as easy as creating a table definition. Below is an example of how to create a new style from an already existing style:

```
proc template;
define style rtftable;
parent=styles.rtf;
  replace fonts /
     'TitleFont2' = ("Times",12pt,Bold Italic)
     'TitleFont' = ("Times",13pt,Bold )
     'StrongFont' = ("Times",10pt,Bold)
     'EmphasisFont' = ("Times",10pt,Italic)
     'FixedEmphasisFont' = ("Courier New, Courier",9pt,Italic)
     'FixedStrongFont' = ("Courier New, Courier",9pt,Bold)
     'FixedHeadingFont' = ("Courier New, Courier",9pt,Bold)
     'BatchFixedFont' = ("SAS Monospace, Courier New,
Courier",6.7pt)
     'FixedFont' = ("Courier New, Courier",9pt)
     'headingEmphasisFont' = ("Times",11pt,Bold Italic)
     'headingFont' = ("Times",11pt,Bold)
     'docFont' = ("Times",10pt);

    replace output/
     rules = ALL
     cellpadding = 4pt
     cellspacing = 0.25pt
     borderwidth = void;

  replace colors
     "Abstract colors used in the default style" /
     'headerfgemph' = #8B0000
     'headerbgemph' = #ffffcc
     'headerfgstrong' = #8B0000
     'headerbgstrong' = #ffffcc
     'headerfg' = #8B0000
     'headerbg' = #ffffcc
     'datafgemph' = black
     'databgemph' = white
     'datafgstrong' = black
     'databgstrong' = white
     'datafg' = black
     'databg' = white
     'batchbg' = color_list('bg')
     'batchfg' = color_list('fg')
     'tableborder' = color_list('fg')
     'tablebg' = color_list('fg')
     'notefg' = color_list('fg')
     'notebg' = color_list('bg')
     'bylinefg' = color_list('fg')
     'bylinebg' = color_list('bg')
     'captionfg' = color_list('fg')
     'captionbg' = color_list('bg')
     'proctitlefg' = color_list('fg')
     'proctitlebg' = color_list('bg')
     'titlefg' = #8B0000
     'titlebg' = #ffffcc
     'systitlefg' = color_list('fg')
     'systitlebg' = color_list('bg')
     'Conentryfg' = color_list('fg')
     'Confolderfg' = color_list('fg')
     'Contitlefg' = color_list('fg')
     'link2' = color_list('link')
     'link1' = color_list('link')
     'contentfg' = color_list('fg')
     'contentbg' = color_list('bg')
     'docfg' = color_list('fg')
     'docbg' = color_list('bg');


end;
run;
```

Like the table definition, you need to give the style you are creating a name. You will then reference this style when you open your ODS destination. When using the parent= option, your new style will contain all the aspects of the parent style except for the options that you change within the TEMPLATE procedure.

In order to see all the individual style components that make a up a particular style, the following code can be used to produce a list within the log file:

```
proc template;
source styles.rtf;
end;
run;
```

The list contained within the log file will look similar to the code above where I changed some of the options. Any of the components listed can be replaced.

Styles can be used across all ODS destinations. Once you have created the style definition, you can use it for all the files that you are creating for that document, regardless of file size.

Macro-*tize* It
The process of making a table definition for each file can be a tedious process. Using options available in the TEMPLATE procedure, do loops and a little trickery, a macro can be created that will create the definition.

The first step is to create a dataset that contains only the columns that you want displayed. This dataset should look exactly as you would like your electronic file to look. The labels for the variables are going to be the column headers and the variables should be formatted.

The next step is to create one macro variable that contains the number of columns in your dataset, one that contains the number of titles and one for the number of footnotes. Additionally, one macro variable is needed for each title and footnote. The macro variables for the titles and footnotes are going to be entered into the table definition as MVARs.

The third step is to create your define statements for columns, headers, and footers. If all headers and footers can be the same font and same justification, you can create one definition and place a do loop around it using the macro variables created in step two. This also applies to the column definitions.

The forth step is to create a style based on your needs. If the standard SAS® styles are appropriate for your output then this step can be skipped.

The fifth and final step is to create your data component and output the object using a combination of set, file and print, and put statements. This step binds your data to your template and style components and resolves your macro variables. An example of these statements is:

```
ods rtf body="c:\paper.rtf" style= rtftable;

 data _null_;
  set build.paper1;
  file print ods=(template='means.inveresk'
                columns=(col1=col1
col2=col2(generic=on) col2=col3(generic=on)
col2=col4(generic=on) col2=col5(generic=on)));
  put _ods_;
  run;
 ods rtf close;
```

Conclusion
Although the TEMPLATE procedure can be a tedious process, it can be made simple to fit individual need. Through the macro process, one template can be used on multiple outputs across multiple ODS destinations.

Contact Information

Cindy Stroupe
Inveresk Research Inc
P.O. Box 13991
Research Triangle Park, NC 27709
Work Phone: (919) 462-2670
Fax: (919) 462-2773
Email: cstroupe@inveresk.com