

My ODS – Real-World Uses of Modifying Table Templates

Steve James, Centers for Disease Control and Prevention, Atlanta, GA

ABSTRACT

Up until SAS® Version 8, you pretty much had to take procedure output the way the SAS System gave it to you. If you wanted it to look different you had to take the extra steps of storing the output somewhere and then modifying it to suit your needs. And you had to do this in every program that used the procedure. Using ODS you now can have modified procedure output without the extra steps. This is done by modifying the table template associated with the procedure. You modify the table template once and the output is the way you define it every time you use it.

This paper demonstrates three examples of changing procedure output. It shows how to make PROC FREQ display frequency counts with commas for frequencies over 1,000 as well as a percentage sign for the percentages. The paper will also show two ways to consolidate PROC CONTENTS output. For a simple change, just one statement limits the output to the list of variables, without having to show the dataset attributes and engine/host information—extras that are often ignored by users of this PROC. A more involved technique combines selected dataset, engine/host and variable attributes information into one table. Though your concerns may differ, the techniques learned here can be adapted to other procedures and other needs.

INTRODUCTION

Before the introduction of the Output Delivery System (ODS), procedure output was controlled by the individual procedure. The data and its presentation were integrated in such a way that you couldn't change the procedure output without changing the procedure itself. Now most¹ Base SAS Software procedures separate data content from its presentation. They create one or more output objects each of which consists of a data component and a table template. By modifying the table template you can change the output format. This paper will discuss several ways of doing this.

ODS CONCEPTS

Before we go further let's cover two important ODS concepts: the output object and table templates.

OUTPUT OBJECTS

Most Base SAS System procedures create one or more output objects. In order to modify procedure output we first have to know what output object it produces. You can get the answer a number of different ways. The simplest is in the windowing environment where you can look in the results window and see the 3 objects and what they're named. If you right-click on any of them you'll see the name of the output object as well as the table template it uses (more on that in a moment). Alternatively you can use the ODS TRACE statement as in the following example:

```
ods trace on / listing ;
proc contents data=sashelp.class ; run ;
ods trace off ;
```

If you were to do the above the output in the listing output would contain the following:

```
Output Added:
-----
Name:      Variables
Label:     Variables
Template:  Base.Contents.Variables
Path:     Contents.DataSet.Variables
-----
```

This tells us that one of the output objects is called VARIABLES and it uses the table template called Base.Contents.Variables. While we can't change the fact that PROC CONTENTS creates the output object VARIABLES nor can we change its name to something like MYVARS, we can modify how the output object looks. Its display characteristics are defined in its table template.

¹ Not all Base SAS Software Procedures that produce output currently have a table template. Those that do not are called the "gnarly four" and are PROC PRINT, PROC FREQ multi-way frequencies (e.g. age*sex) without the LIST option, PROC REPORT and PROC TABULATE. Their complexity makes defining a table template difficult.

TABLE TEMPLATES

What is a table template? You can think of a table template as the code (or model or template) that tells the SAS software how to create an ideal output object. It's a set of instructions if you will. Output objects are created using the instructions stored in a table template, so if you want to change the procedure output, you have to modify the table template used to create the output object. Since most of SAS procedure output is tabular in structure, each procedure writer was responsible (between SAS 6 and SAS 7) for converting how his or her procedure would use a table template. Column order, column formats, and column headers, are just some of the things controlled by a procedure's table template. Think of the table template as being the "ideal" definition for how a particular procedure should create its tabular output. Then when the procedure is invoked using an actual SAS data set, the procedure binds the data (for this one run) to the ideal template for the procedure (from the template store), takes any particular procedure options into account and produces a specific output object based on the data from this run. The next time the procedure is invoked, the same "ideal" model or template is bound to this new data and a new output object table is produced.

Table templates are stored in a SAS bitmapped file called a template store. These are files with an extension of .sas7bitm. SAS Institute supplies a template store with all the table templates in a store called TMPLMST located in the SASHELP directory. So, for instance, yours might be in
c:\program files\SAS\The SAS System\9.1\SASHELP\tmplmst.sas7bitm.

However you don't ever want to modify the table templates in SASHELP.TMPLMST. Rather, you place modified versions of the table templates in your own template store located separately from the one SAS provides. You'll want to store all your modified versions of these separately from those SAS supplies so that you can always go back to the starting point if necessary.

You create your own template store simply by defining a writable item store location and putting something in it. When you create a table template using PROC TEMPLATE, SAS software stores the modified template in the first template store it sees that has update capability. The order that the software looks at template stores is defined by the ODS PATH statement. Consider the following:

```
ODS PATH SASUSER.MYTEMPLATES (UPDATE) SASHELP.TMPLMST (READ) ;
```

This instructs the SAS software to first look in a template store called MYTEMPLATES in my SASUSER directory and then look into the TMPLMST store stored in SASHELP. The UPDATE parameter tells the SAS software that the template store can be updated and the READ parameter says that it can only be read from. You always want to specify the READ parameter on the SASHELP.TMPLMST template store so you don't inadvertently overwrite the SAS-supplied table templates. However the Windows® operating system does not honor the READ parameter and you will want to prevent the template store from being modified at the operating system level by using something like the read-only attribute for the template store. The default ODS PATH is

```
ODS PATH SASUSER.TEMPLAT (UPDATE) SASHELP.TMPLMST (READ) ;
```

You may choose to store all your table templates in your SASUSER.TEMPLAT template store and not bother with the ODS PATH statement at all. Or while you're first developing your table templates, you may even choose to create a template store in your WORK directory, which would be in effect only until you exit the SAS system. For instance:

```
ODS PATH WORK.TEMP (UPDATE) SASUSER.MYTEMPLATES (UPDATE) SASHELP.TMPLMST (READ) ;
```

To see what template stores contain, most people will use the SAS windowing environment. Select View→Results Window, and then right-click on the word "Results" in the Results window. Then select "Templates" and a window will open and show you the template stores that are defined in your ODS Path statement (empty ones will not display however). You can expand the templates and see what they contain.

For those who want a programming solution, you can use PROC TEMPLATE. For example :

```
proc template ;  
  list / store=sashelp.tmplmst ;  
  list base /store sashelp.tmplmst ;  
  list base.freq / store=sasuser.mytemplates ;  
run ;
```

SAMPLE DATA SET

In order to have a dataset that will demonstrate some of the features of modifying table templates, we will use the following SAS code:

```
data sasuser.myclass ;
  set sashelp.class ;
  multiplier = 1234 ; * some arbitrary large number ;
  label sex = 'Gender' weight='Weight in lbs' multiplier='Observation Multiplier' ;
  format sex $1. weight 5.1 ;
  informat sex $1. ;
run ;
```

This is the dataset that will be referred to for the remaining of this paper.

MY PROC CONTENTS – SIMPLE VERSION

Often when you want to see the contents of a SAS dataset, you just want to see what variables are in it. You may not care about some of the information included by default. You just want to see the list of variables.

Standard PROC CONTENTS output is shown in Appendix 1. What we want the output to be is just the list of variables, the third of the three tables.

As we stated earlier, SAS procedures create one or more output objects. The Contents procedure creates three output objects: Attributes, Engine/Host, and Variables. We want to tell the SAS software not to display the information containing SAS Dataset Attributes and the Engine/Host information. Using ODS TRACE described above, we can determine that the three output objects that PROC CONTENTS is producing are called ATTRIBUTES, ENGINEHOST and VARIABLES. So if we only want to see the list of variables, we need to suppress the output of the Attributes and the EngineHost output objects. We're not preventing them from being created, but we're going to exclude them from the output.

```
ODS EXCLUDE Attributes EngineHost ;
```

Alternatively we could have simply selected the variables output object with the command:

```
ODS SELECT VARIABLES ;
```

The default behavior is for this statement to be in effect only until the next procedure it encounters. If you want the behavior to continue, use the PERSIST option. To tell the system to continuously exclude these two objects from the output, add the PERSIST option as follows:

```
ODS EXCLUDE Attributes (persist) EngineHost (persist) ;
```

This is something that some may choose to put into an AUTOEXEC.SAS file so that the two objects are always excluded. However if more than one person uses the same autoexec file, this could lead to confusion so use this with caution.

To return to the default behavior, issue this command:

```
ODS SELECT ALL ;
```

MY PROC FREQ

If you run the following PROC FREQ code, you will get the results in Figure 1

```
proc freq data=sasuser.myclass ;
  tables sex ;
  weight multiplier ;
  title 'Standard Proc Freq' ;
  footnote "Standard Proc Freq.sas" ;
run ;
```

Standard Proc Freq

The FREQ Procedure

Gender				
Sex	Frequency	Percent	Cumulative Frequency	Cumulative Percent
F	11106	47.37	11106	47.37
M	12340	52.63	23446	100.00

Standard Proc Freq.sas

Figure 1.

Notice that you have to mentally insert a comma in the frequency counts. You also have to remember that the percentages are actually percentages. Wouldn't it be nice to have commas in the frequencies and percent signs with the percentages? You can do this easily by modifying the table template for PROC FREQ.

Using ODS TRACE as described above, you can determine that the output object for a one-way PROC FREQ is the Base.Freq.OneWayFreqs² table template. There are two ways to get the source code for this table template. Again most will use the windowing environment and simply drill down into SASHELP.TMPLMST and select the table template that you want. You would then copy the code and then put it in the Enhanced Editor. Another method is to use PROC TEMPLATE as follows:

```
proc template;
source base.freq.OneWayFreqs / store=sashelp.tmplmst file='c:\My ODS\OneWayFreqs.sas' ;
run ;
```

Note: "Proc Template;" and "Run;" must be added to code obtained from the above PROC TEMPLATE code.

In this case we simply want to change the format in which the data are output. We'll use the EDIT command of Proc Template. Consider the following code snippet:

```
proc format ; picture picpct low-high ='009.99%';

proc template;
  edit Base.Freq.OneWayFreq;
  column Line FVariable Variable FListVariable ListVariable Frequency TestFrequency
          Percent TestPercent CumFrequency CumPercent ;
  edit Frequency ;
    format = comma10. ;
  end ;
  edit Percent;
    format = picpct. ;
  end;

etc.
```

² The same technique will show that Base.Freq.List is the table template for a multi-way frequency using the LIST options (e.g. tables age*sex / list ;). Further delving into the code will show that Base.Freq.OneWayList is used by both Base.Freq.OneWayFreqs and Base.Freq.List. Edit Base.Freq.OneWayList to get the same effect on both the one-way and multi-way frequencies with LIST option output. The fact that two different invocations of PROC FREQ can both access a single template and inherit some of the specifications from that template and other specifications for specific options from a different template illustrates how powerful and rewarding the study of table templates can be.

Because we're using the EDIT command, all the attributes that we don't define are inherited from the parent Base.Freq.OneWayList that's found in SASHELP.TMPLMST. For instance, the variable Frequency will retain the default header and justification. You'll want to make similar changes for CumFrequency and CumPercent. The complete source code is shown in Appendix 2. It should be noted that in using this technique the picture format must be available whenever Proc Freq is used, not just when creating the table template. As such it is a good candidate to put in an AUTOEXEC.SAS file or store in a formats catalog that you use regularly.

After modifying the template, if you run the following code you'll get the output in Figure 2.: Notice the line with "The Freq Procedure" has been removed. That's accomplished by the ODS NOPROCTITLE command.

```
ods noproctitle ;
proc freq data=sasuser.myclass ;
tables sex ;
weight multiplier ;
title 'My Proc Freq' ;
footnote "My Proc Freq.sas" ;
run ;
```

My Proc Freq

Gender				
Sex	Frequency	Calc Percent	Cumulative Frequency	Cumulative Percent
F	11,106	47.4%	11,106	47.4%
M	12,340	52.6%	23,446	100.0%

My Proc Freq.sas

Figure 2.

It should be noted that you cannot change the name of the table template. You cannot, for instance, change the name of the table template to MyOneWayFreqs. The name is "hard-coded" into Proc Freq and it will only look for the one name. However what you can do is to change where the procedure finds the table template. You do that using the ODS PATH statement described above. Consider the following ODS PATH statement:

```
ODS PATH SASUSER.MYTEMPLATES (UPDATE) SASHELP.TMPLMST (READ) ;
```

ODS will now use the modified template in SASUSER.MYTEMPLATES to produce the output. However at this point only the output of PROC FREQ is modified and in addition it's only one-way frequencies that are modified. For all the other procedures it's still using the table templates from SASHELP.TMPLMST. The reason is the software is now using the new template because of the way the ODS PATH statement is defined. After running the template code, your two template stores look like this:

SASUSER.MYTEMPLATES	SASHELP.TMPLMST
Base.Freq.OneWayList	BASE.CATALOG.RANDOM Base.Catalog.Random Base.Catalog.Sequential ... Base.Freq.OneWayList ...

When the SAS software begins looking for a table template, it starts processing the template stores listed in the ODS PATH statement, going from left to right, until it finds the item it's looking for. The fact that there are two Base.Freq.OneWayList table templates in the path is not a problem because the software simply uses the first one it encounters, which is in SASUSER.MYTEMPLATES in this case.

If you want to return to the default behavior for PROC FREQ, you can remove the modified table template permanently with the following code:

```
proc template ;
  delete base.freq.OneWayList / store=sasuser.mytemplates ;
run;
```

Or if you just want to temporarily change the behavior you can ensure that the SAS software uses the default table template by changing to ODS PATH statement to the following:

```
ODS PATH SASHELP.TMPLMST (READ) SASUSER.MYTEMPLATES (UPDATE) ;
```

Remember, the software uses the first object that it encounters based on how things are defined in the ODS PATH statement, going from left to right.

MY PROC CONTENTS – VERSION TWO

For something a little more challenging, our next task will be to modify the appearance of PROC CONTENTS so that it consolidates the information into one table. Standard PROC CONTENTS output is shown in Appendix 1. However, suppose we want to combine that information so that it's more concise? One example of doing that is in Figure 3.

My Proc Contents

SAS Data Set = sasuser.myclass			
Variables = 6	Obs = 19		Label
Variable Name	Type:Length	Format Informat	
Age	Num : 8		
Height	Num : 8		
Name	Char : 8		
Sex	Char : 1	\$1. \$1.	Gender
Weight	Num : 8	5.1	Weight in lbs
multiplier	Num : 8		Observation Multiplier
Host Name: \\fsp-corpdata1\home\spj1\docs\My SAS Files\9.1\myclass.sas7bdat			
Last Modified: Tuesday, June 20, 2006 05:53:56 PM			

My Proc Contents.sas

Figure 3.

Notice what the output contains here. It contains all of the information about the variables that's in the default output plus it has the name of the SAS dataset, the number of variables, the number of observations, the last modified date and the host file name. The additional information that we want is produced by PROC CONTENTS, but it's produced in output objects that we don't want to display. However if we simply grab those few pieces of information from the objects and add them to what we want displayed, we get everything we want and nothing we don't want. The code to do this is in Appendix 3. An explanation of some of the key points follows.

CREATE A TABLE TEMPLATE WITH EMBEDDED MACRO VARIABLES

The first thing is to create a template for displaying the output the way we want it. The table template for the Variables output object doesn't come with places for things like the dataset name and the number of variables. We have to create those ourselves. What we'll do is to create headers and footers for the additional information. We can create a header with the following code:

```
header h1 ;
define h1 ;
    text "text for header" ;
end ;
```

And similarly a footer with the following code:

```
footer f1 ;
define f1 ;
    text "text for footer" ;
end ;
```

That's how the additional headers and footers are defined but how do we get the data that we want in there. In this example for instance, we want the name of the SAS dataset in the header. We'll use macro variables. We'll show you how to put values in those macro variables shortly. But the first thing is to properly put them in the template so that they display correctly. However our first inclination on how that might work has some problems. We might think the way to code it would be the following:

```
define h1 ;
    text "Data Set Name:" &data ;
end ;
```

The problem with this approach is that the macro variable *data* would resolve at the time the table template was created not when the table template was used. To overcome this problem SAS developed the MVAR statement to identify macro variables that will be resolved when the table template is used, not when it is created. Here's the way the code would look:

```
mvar data ;
define h1 ;
    text "Data Set Name:" data ;
end ;
```

The MVAR statement tells the software that data is a macro variable and it should get its value from the macro symbol table at execution time, not at compile time. Note that DATA does not have an ampersand at the front.

OTHER TEMPLATE MODIFICATIONS

If you look at Figure 3, you'll notice that the output that we want has three extra headers (two on one line) and two footers in the table. Those are defined in the table template definition. Consider the following code snippet:

```
proc template ;
    define table Base.Contents.Variables;
        column Variable Len Type TypeLen Pos Flags (Format Informat) Label Transcode;
        mvar data nobs nvars;
        header h1 h2 h3;
        define h1;
            text "SAS Data Set = " data;
            space = 1;
        end;
        define h2;
            text "Variables = " nvars;
            space = 1;
            start = Variable;
        end;
    enddefine;
endproc;
```

```

        end = Len;
    end;
    define h3;
        text "Obs = " nobs;
        space = 1;
        format = comma12. ;
        start = TypeLen;
        end = Format;
    end;
    etc...

```

The statement

```
header h1 h2 h3 ;
```

tells the SAS software that we want to define 3 cells that will be the headers in the table. The default behavior is for the cell to span the entire table, which is done for h1. However for h2, we only want it to show above the column Variable. So we designate the starting column (start=Variable) and an ending column (end=Len). It's not important that Len does not display in the output.

We're also creating a user-defined column called TypeLen. It's the concatenation of the Type and Len columns and saves space by putting them both into the same column. Additionally the Format and Informat are stacked in one column. This is done by enclosing the column names in parentheses.

We need only run the Proc Template code once as we're simply creating a modified table template similar to what we did earlier with Proc Freq. The difference is that to fill the table template with all of its fields, we need to run additional code to populate the macro variables that are defined in the headers and footers. Without it the appropriate macro variables will not be defined and the table template will have missing information in it.

CREATE MACRO VARIABLES WITH THE REQUIRED INFORMATION

To get the information that we want from the output objects is mostly standard SAS programming with one exception. We need to convert the information in the output object into SAS datasets so we can access it. Once done we'll use a DATA STEP to read the information and use CALL SYMPUT to store the values in the macro variables. The code to store the contents of the output object in a SAS dataset is the ODS OUTPUT statement. For example:

```
ODS OUTPUT ATTRIBUTES = WORK.ATTRIBUTES ;
```

Note that this statement comes before the PROC CONTENTS statement. It is saying in essence "When the output object named Attributes is created, store its contents in WORK.ATTRIBUTES." This is different from "Convert the output object named Attributes into WORK.ATTRIBUTES." The issue is timing.

Once the information from the output objects have been stored in a SAS dataset, you can then use standard SAS tools such as a DATA STEP or PROC SQL to store the appropriate values into macro variables.

EXECUTING THE PROC CONTENTS

Because we have to create the macro variables each time the procedure is called, we can't simply run a PROC CONTENTS and have the output appear as it does in Figure 3. We must also run the SAS code that creates the macro variables. The most reasonable way to do this is with a macro. The complete code for this is in Appendix 3.

The code contains two parts. The first part modifies the table template for PROC CONTENTS and needs to be run only once. The macro needs to be run every time. However even after you modified the table template you can still run a simple PROC CONTENTS as before without using the macro. The only difference would be that your output would have missing values where the macro variables would have been. If you want to have the default PROC CONTENTS just as it's shipped from SAS Institute, you would issue the following two commands:

```
ODS SELECT ALL ;
ODS PATH SASHELP.TMPLMST(READ) ;
```

CONCLUSION

Modifying table templates gives you a never-before ability to change procedure output straight from the procedure. While it may seem a daunting proposition at first, with proper care you can accomplish some very useful things. The three examples here are useful in themselves, but more importantly they show the general types of things that can be done and provide an example of how a user can make changes to table templates to solve his or her own specific needs.

REFERENCES

Advanced Output Delivery System Topics – Course Notes, SAS Institute.

ACKNOWLEDGMENTS

Many thanks to Cynthia Zender of SAS Institute for providing her technical expertise, without which this paper would not have been possible.

CONTACT INFORMATION

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

Steve James
Centers for Disease Control and Prevention
1600 Clifton Road NE, MS-E57
Atlanta, Ga. 30333

Work Phone: (404) 639-6041
Fax: (404) 639-8555
Email: sjames@cdc.gov

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.

Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

APPENDIX 1

Standard Proc Contents

The CONTENTS Procedure

Data Set Name	SASUSER.MYCLASS	Observations	19
Member Type	DATA	Variables	6
Engine	V9	Indexes	0
Created	11:00 Sunday, June 5, 2005	Observation Length	48
Last Modified	11:00 Sunday, June 5, 2005	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS		
Encoding	wlatin1 Western (Windows)		

Engine/Host Dependent Information	
Data Set Page Size	4096
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	84
Obs in First Data Page	19
Number of Data Set Repairs	0
File Name	c:\My SAS Files\myclass.sas7bdat
Release Created	9.0000M0
Host Created	XP_HOME

Alphabetic List of Variables and Attributes						
#	Variable	Type	Len	Format	Informat	Label
3	Age	Num	8	2.		
4	Height	Num	8			
1	Name	Char	8			
2	Sex	Char	1	\$1.	\$1.	Gender

Alphabetic List of Variables and Attributes						
5	Weight	Num	8			Weight in lbs
6	multiplier	Num	8			Observation Multiplier

Standard Proc Contents.sas



APPENDIX 2

```
/*-----
This program shows how to modify the table template for PROC FREQ so
that frequencies appear with a comma10. format and percentages appear
with a percent sign (%).
-----*/
ods path work.mytemplates (update) sashelp.tmplmst (read) ;

/*-----
A PICTURE format is used to display the percentages with the percent sign and must
be available whenever PROC FREQ is used. Put in AUTOEXEC.SAS to ensure it's there.
-----*/
proc format;
  picture picpct low-high = '009.99%';
run;

proc template;
  * delete any old copies so you don't EDIT them rather from SASHELP.TMPLMST ;
  delete base.freq.OneWayList / store=sasuser.mytemplates ;

  edit Base.Freq.OneWayList;
  notes "My Base.Freq.OneWayFreqs" ;
  column Line FVariable Variable FListVariable ListVariable Frequency TestFrequency
         Percent TestPercent CumFrequency CumPercent ;
  edit Percent;
    format=picpct. ;
  end;
  edit Frequency ;
    format = comma10. ;
  end ;
  edit CumFrequency ;
    format = comma10. ;
  end ;
  edit CumPercent;
    format=picpct. ;
  end;

  end;
run;

ods rtf file='My Proc Freq.rtf' style=rtf bodytitle ;

ods noproctitle ;
proc freq data=sasuser.myclass ;
tables sex ;
weight multiplier ;
title 'My Proc Freq' ;
footnote "My Proc Freq.sas" ;
run ;

ods rtf close ;
```

APPENDIX 3

```
/*-----
This program modifies the Table Template for Proc Contents so that it
contains some Data Set Attribute information in the table with the
Variable information. It must be run as a macro so that the appropriate
macro variables are defined prior to their being used in the table template.
-----*/

ods path work.mytemplates (update) sashelp.tmplmst (read) ;

/*-----
Define the Table Template for Proc Contents using the new format. This
PROC TEMPLATE code need be run only once.
-----*/

proc template;
define table Base.Contents.Variables;
  notes "Contents Variables";
  dynamic name_width name_width_max label_width label_width_max;
  mvar data nob nvars modate file_name ;
  column Variable Len Type TypeLen Pos Flags (Format Informat) Label
Transcode;
  header h1 h2 h3;
  define h1;
    text "#SAS Data Set = " data;
    space = 1;
  end;
  define h2;
    text "#Variables = " nvars;
    space = 1;
    start = Variable;
    end = Len;
  end;
  define h3;
    text "#Obs = " nob;
    space = 1;
    format = comma12. ;
    start = TypeLen;
    end = Format;
  end;
  footer f1 f2;
  define f1;
    text "#Host Name: " file_name ;
    space = 1;
    just=left ;
    spill_adj;
    spill_margin;
  end;
  define f2;
    text "#Last Modified: " modate ;
    space = 1;
    just=left ;
    spill_adj;
    spill_margin;
  end;
  define Variable;
    header = "#Variable#Name";
    width_max = name_width_max;
    width = name_width;
    just = 1;
  end;
end;
```

```

define Type;
    header = "#Character#or Numeric";
    just = c;
    print=off ;
end;
define Len;
    header = "#Length";
    just = c;
    print=off ;
end;
    define typelen ;
        header = "#Type:Length" ;
        compute as put(type,$char7.)||':'||put(len,comma6.);
    end;
define Pos;
    header = "Pos";
    print = OFF;
end;
define Flags;
    header = "Flags";
end;
define Format;
    header = "#Format#Informat";
    just = c;
end;
define Informat;
    header = "Informat";
    just = c;
end;
define Label;
    header = "Label";
    width_max = label_width_max;
    width = label_width;
    flow;
end;
define Transcode;
    header = "Transcode";
end;
col_space_max = 4;
col_space_min = 1;
newpage = OFF;
center;
balance;
end;
run ;

/*-----
This macro produces a PROC CONTENTS-like output that also contains
additional dataset information. It uses a modified table template
that uses macro variables that this macro creates.
-----*/

%macro mypc(data=) ;
%local modate nob nvars dsn ;

ods exclude Variables ; * exclude Variable list from first Proc Contents;
ods exclude Attributes EngineHost ;

ods output Attributes=work.attributes ; * Store Attributes in SAS dataset ;
ods output EngineHost = work.enginehost ;

```

```

/*-----
Create Proc Contents object so that we can obtain the
Attribute and EngineHost information.  But do not display
any of the output for this step.
-----*/
proc contents data=&data ;
run ;

/*-----
Read the attribute information and store it in macro variables that can
be used by the table template.
-----*/
data _null_;
  set work.attributes;
  if _n_=1 then call symput('nobs',cValue2);
  else if _n_ = 2 then call symput('nvars',cValue2);
  else if _n_ = 5 then call symput ('modate',cValue1) ;
run;

/*-----
Now read the EngineHost information and store it in a macro variable.
-----*/
data _null_ ;
  set work.enginehost ;
  if _n_ = 7 then call symput('file_name',cValue1) ;
run ;

%put file_name = &file_name ;

/*-----
Now run the procedure again and produce the output with the macro variables
defined.
-----*/
ods exclude Attributes EngineHost ;

proc contents data=&data ;
run ;

%mend mypc ;

*-----
Now run the macro that produces the Proc Contents Output.
-----*/
ods rtf file='My Proc Contents.rtf' bodytitle ;
title 'My Proc Contents';
footnote "My Proc Contents.sas" ;
%mypc (data=sasuser.myclass) ;
ods rtf close ;

```