Paper SS-02

Misquoting Jane Austen in the Name of Quality Assurance

Deborah Posner, Westat, Rockville, MD

ABSTRACT

It is a truth universally acknowledged
that a SAS® programming task skillfully programmed
must be in search of proper quality assurance.

– Misquoted from *Pride and Prejudice*

A proficient programmer may not be an eager code checker. Have you ever heard a programmer announce that a task was completed, followed by someone else finding errors in the log, unexpected results, or (horrors!) no results at all? Have you ever been that programmer? The reality is that no matter how many times we say checking your work is part of the job, the tasks of programming and checking are intrinsically distinct. Virtuous programmers and their vigilant managers must determine the best way to accomplish the quality assurance task with regard to thoroughness and expedience.

This paper outlines specific quality assurance tasks for programmers to include in their standard workflow. It also provides programming code to create outputs expressly designed for programmers, managers, and specification writers for their review. These outputs report on missing values in the data, missing labels for variables, values (i. e., codes) without labels, and values where labels exist for expected data, but no data occurred for these values.

INTRODUCTION

The output is rich, to be sure, and you have many fine data sets and fine spreadsheets. But will they make your client happy?

– Misquoted from *Pride and Prejudice*

Quality Assurance (QA) includes a review of program outputs, including logs, listings, data sets, and data files in non-SAS® formats. Fidelity to specifications and requirements documents must also be checked. The question is: who is responsible for QA? A specification writer may assume the programmer has performed a thorough quality check, and that same programmer may assume his manager will be doing QA, and that manager "knows" that the programmer and specification writer will catch any problems. In this scenario, everyone should be doing the QA, but no one is actually doing it.

If there are undetected problems, they will surface at a later stage of work, such as the next round of programming, when client deliverables are created, or maybe by the client. There will be a penalty for not doing the QA in line with the programming task.

This paper addresses specific QA tasks for SAS programmers to incorporate in their programming. Many are applicable to every programming task performed. Some are intended for programmers and technical managers, and others facilitate QA for specification writers and other stakeholders. These stakeholders may be non-technical managers, statisticians, or clients. The code will create outputs to answer the following questions:

- Do all variables have labels?
- Are there missing values for any variables? If so, which variables have missing values?
- With variables formatted by the FORMAT procedure…
    - do all values have labels?
    - do all labels get assigned to values, or are there labels for which there is no data?

Unlabeled values may indicate unexpected values in the data. If a value label was created for a value that does not appear in the data set, this too is a "red flag." It may be necessary to investigate outliers or simply tweak the PROC FORMAT's value statement. Although especially helpful in survey data sets, these tools are applicable to all data sets.

This paper also has code for parsing the log. The output can be used as a table for reviewing inputs, outputs, and their observation and variable counts. The parsed log output is also an effective start for a flow chart, which is especially helpful for reviewers who appreciate a graphic display of information.

FOR THE PROGRAMMERS: INDISPENSIBLE QA ITEMS

Good code ceases to be good code if it is done by good programmers in a hurried way.

– Misquoted from *Emma*

There are many important checks to do with any programming task. The following list highlights selected tasks for their importance or efficiency.

- **Have you read the log?** This is an overwhelming task for long or complicated programs. Before taking on the entire log, take 30 seconds to search for the words **error**, **warning**, and **uninitialized**. Errors are always bad news, and warnings often are too. It is unlikely that the term "uninitialized variable" will appear in a log without a programming error nearby. Two common sources of uninitialized variables are spelling errors and omissions on keep statements.
- **Read the specification or requirements – again.** It is easy to focus on the programming and miss output requests – or visa-versa. Check both the code and the outputs against the specification. Save a copy of the specification as a record of the request. If informal changes have been made via email, telephone, or water-cooler caucusing, update the specification. Emails can easily be saved as text files and stored with more formal specifications.
- **Run and review the CONTENTS procedure contents on every permanently saved data set.** PROC CONTENTS is a simple and effective way to document data products; do this even if not specified.
- **Crosstabs: one for each new variable, crossed with its inputs.** Again – this is very important and should be done even if not explicitly requested. No crosstab should be 20 pages: simplify the output of continuous variables by formatting.
- **Print a sample of data set and review.** Looking at the first 50 observations is common but a sample consisting of the first 50 observations will not likely be representative of the entire data set. SAS/STAT users have a very simple solution available: with the SURVEYSELECT procedure, one line of code creates a simple random sample (SRS) that will be better for review.

```
PROC SURVEYSELECT data=inds method=SRS sampsize=50 out=outds;
```

- **Put the program name and its location in a title or footnote.** This simple practice will save hours in the long run.
- **Check the observation counts at every data step.** Unexpected changes in observation counts need to be understood.
- **Labels – lots of labels**. Variables, values, and even data sets. Ask the specification writers to provide labels. No one ever regrets taking the time to create labels, especially after a couple years have elapsed.
- **Missing values – ok or not?** Do you have missing values? Missing values are expected in some instances. If missing variables are not expected, trace their origin by reviewing earlier data steps.

Good documentation is very important but beyond the scope of this paper. Rick Mitchell's 2007 SAS Global Forum paper *Finding Your Mistakes Before They Find You: A Quality Approach For SAS® Programmers* is highly recommended for more QA suggestions.

TOOLS TO AID IN REVIEW

I am in no humour at present to review output
the programmer has not thoroughly reviewed.

-misquoted from *Pride and Prejudice*

The tools presented below will create outputs to help programmers perform their QA as well as facilitating the review of specification writers and other stakeholders. The examples use a data set named Widgets. The data in Widgets and the PROC FORMAT code used to format its variables are in the appendix.

ARE ANY VARIABLE MISSING LABELS?

There are many ways to check for variable labels. This method is simple and effective. If you're not experienced with the SQL procedure, this is also a great place to beginning utilizing this powerful procedure. See more on data cleaning with PROC SQL in Ron Cody's papers and books such as Cody's *Data Cleaning Techniques Using SAS Software*.)

```
PROC SQL;
  select name label="Variable Name"
    from dictionary.columns
    where libname = "WORK"          /* specify the libname */
    and memname = upcase("Widgets") /* check this data set */
    and memtype = "DATA"
    and label=" ";                  /* missing label? */
quit;
```

ARE THERE MISSING VALUES FOR ANY VARIABLES?

There may be times when missing values are unacceptable in a data set. For example, a specification for a questionnaire data set may require all missing values be categorized as refused, unknown, or not asked. Scanning frequencies is one way to find missing values, but this solution's effectiveness is limited by the number of variables in a data set.

The following code solution creates a list of variables with missing values and the count of missing values. This code is easy to implement and much faster than reading the frequencies of a 2,000 variable data set…or even a 200 variable data set!

Using the Widgets data set, we create two arrays: one of all character variables and one of all numeric variables. On each observation, we check each character variable's value; if missing the variables name is captured with the vname function and output to the ListMissing data set. The same is done for the numeric variables. After the Widgets data set is processed, we run PROC SQL to summarize the ListMissing data set. Instead of var appearing on two rows, there is now one row for var and a count of 2.

```
DATA ListMissing;
  set widgets;
  array c _char_;
  array n _numeric_;
  do over c;
    if c="" then do;
      var=vname(c);
      output;
    end; /* if c="" then do*/
  end; /* do over c */
  do over n;
    if n=. then do;
      var=vname(n);
      output;
    end; /* if n=. then do */
  end; /* do over n */
  format _all_;
run;

title "Variables with Missing Values";
PROC SQL;
  select distinct var length 32,
    count(*) as n
    from ListMissing
```

```
    order by var;
run;
```

var	n
instock	2

VALUES WITHOUT LABELS AND LABELS WITH VALUES

Some data sets require a label for every value. A survey may have coded responses and PROC FORMAT is the perfect tool for associating the codes (values) with their labels. Labels are likely created during the survey development process and may not be subsequently updated. A macro that identifies unlabeled values is valuable in preparing data sets for analysis and delivery.

There may times when it is helpful to know about value labels that exist in the PROC FORMAT value statements but are never used. This macro will identify these values also.

The macro vallmss has two parameters: **DSName** for the data set name that will be checked and drop for a list of variable that will not be checked. **drop** should include all variables for which no format is assigned through a PROC FORMAT's value statement. Details of the macro's operations are in the macro code below with each data step and proc.

```
%macro vallmss(DSName,drop);

/* Create a list of variable and value pairs for character variables */
DATA ActualValues1c(keep=var value);
  length var value $32;
  set &dsname.(drop=&drop.);
  array c _char_;
  do over c;
    var=vname(c);
    value=strip(c);
    output;
  end; /* do over c */
run;

/* Create a list of variable and value pairs for numeric variables */
DATA ActualValues1n(keep=var value);
  length var $32;
  set &dsname.(drop=&drop.);
  array n _numeric_;
  do over n;
    var=vname(n);
    value=n/1;
    output;
  end; /* do over n */
run;

/* Export the format catalog to a data set */
PROC FORMAT library=work
  cntlout=FmtDetail1(keep=fmtname start end type label fuzz);
run;

/* Subset the format data set to character and number format data sets */
DATA
  FmtDetail2c(keep=fmtname start label rename=(start=value))
  FmtDetail2n(keep=fmtname start_n end_n rename=(start_n=start end_n=end));
  set FmtDetail1;
  start=strip(start);
  end=strip(end);
  if type="C" then output FmtDetail2c;
  else if type="N" then do; /* Transform the start and end ranges to number */
    start_n=start/1;
    end_n=end/1;
    output FmtDetail2n;
  end; /* if type="N" */
run;
```

```
PROC SQL;
/* Create a list of variables and their format assignments */
create table VarsWithFormats as
  select name, format
  from dictionary.columns
  where libname = "WORK"
  & memname = upcase("&DSName.")
    & memtype = "DATA"
    & format^=" ";

/* Deduplicate the list of variable and value pairs for numeric variables */
create table ActualValues2n as
  select distinct var, value
  from ActualValues1n;

/* Add format name to the list of numeric variable and value pairs */
create table ActualValues3n as
  select ActualValues2n.*, compress(format,".") as format
  from ActualValues2n
  inner join VarsWithFormats
  on ActualValues2n.var = VarsWithFormats.name;

/* Create a list of numeric variable and value pairs if the value is formatted */
create table ValuesWithFormatsn as
  select ActualValues3n.*
  from ActualValues3n
  inner join FmtDetail2n
  on ActualValues3n.format = FmtDetail2n.fmtname
  where start <= value <= end;

/* Output a list of numeric variable and value pairs with no label for the value */
title "Numeric Variables with Unlabeled Values";
select ActualValues3n.*
  from ActualValues3n
  full outer join ValuesWithFormatsn
  on ActualValues3n.var = ValuesWithFormatsn.var
  & ActualValues3n.value = ValuesWithFormatsn.value
  where ValuesWithFormatsn.var is null;
```

var	value	format
color	8	COLORF

```
/* Output a list of numeric format values that were not assigned */
title "Numeric Format Values with Labels Not Assigned to Any Data";
select FmtDetail2n.*
  from ActualValues3n
  full outer join FmtDetail2n
  on ActualValues3n.value = FmtDetail2n.start
  where ActualValues3n.var is null;
```

start	end	label	format
1	1	Red	COLORF
2	2	Orange	COLORF
6	6	Indigo	COLORF

```
/* Deduplicate the list of variable and value pairs for character variables */
create table ActualValues2c as
  select distinct var, value
  from ActualValues1c;

/* Add format name to the list of character variable and value pairs */
create table ActualValues3c as
  select ActualValues2c.*, compress(format,"$.") as format
  from ActualValues2c
  inner join VarsWithFormats
```

```
  on ActualValues2c.var = VarsWithFormats.name;

/* Create a list of character variable and value pairs if the value is formatted */
create table ValuesWithFormatsc as
  select ActualValues3c.*
  from ActualValues3c
  inner join FmtDetail2c
  on ActualValues3c.format = FmtDetail2c.fmtname
  where ActualValues3c.value = FmtDetail2c.value;

/* Output a list of character variable and value pairs with no label for the value */
title "Character Variables with Unlabeled Values";
select ActualValues3c.*
  from ActualValues3c
  full outer join ValuesWithFormatsc
  on ActualValues3c.var = ValuesWithFormatsc.var
  & ActualValues3c.value = ValuesWithFormatsc.value
  where ValuesWithFormatsc.var is null;
```

var	value	format
Size	L	SIZEF

```
/* Output list of character format values that were not assigned */
title "Character Format Values with Labels Not Assigned to Any Data";
select FmtDetail2c.*
  from ActualValues3c
  full outer join FmtDetail2c
  on ActualValues3c.value = FmtDetail2c.value
  where ActualValues3c.var is null;
```

start	label	format
XS	Extra Small	SIZEF

```
quit;

%mend; /*** macro vallmss ***/
```

SUMMARIZING THE PROCESSING BY PARSING THE LOG

This code will parse SAS logs, providing a tabular summary of data step and procs. It is especially helpful for checking the observation and variable counts for input and output data sets and for a creating flowchart that graphically summarizes a program's steps. This macro is easy to run and can be modified to highlight other log issues of specific interest to the programmer.

Although this code is limited to merge and set statements with two data sets, it can be modified to account for more data sets in one statement.

This code was adapted from Michael Raithel's SUGI 30 paper *Programmatically Measure SAS® Application Performance On Any Computer Platform With the New LOGPARSE SAS Macro*.

```
options center=no mprint mlogic symbolgen;

/* parameters lib is the library where log is stored; log is log name */
%macro Parse(lib,log);
data parse1;
  infile "&lib.&log." lrecl=250 end=lastrec pad truncover;
  input line $char200.;
  line = upcase(line);
  /* assign a row number for each line in the log */
  row=_n_;
  /* delete rows without processing information */
  if compress(line)=" " or
    substr(line,1,1) = '0c'x or
    index(line, 'THE SAS SYSTEM') >0 or
    compress(line,'0123456789') = " "
    then delete;
run;
```

```
data parse2;
  length row 3 linetype $4;
  set parse1;
  /* use lag function to get content of previous (lagging) line */
  line_lag1=lag1(line);
  /* mark some lines as code */
  if compress(scan(line,1,' '),':0123456789') = " " then linetype="code";
run;

data parse3 (drop=mergeflag byflag line line_lag1 linetype row);
  length stepname $20 step mergeflag  byflag 3 infolder1 $8 inds1 $32 inobs1 8 infolder2
$8 inds2 $32 inobs2 8 outfolder $8 outds $32 outobs outvars 8;
  set parse2;
  /* use retain to maintain values while processing a data step or proc */
  retain stepname mergeflag byflag infolder1 inds1 inobs1 infolder2 inds2 inobs2
outfolder outds outobs outvars;
  /* assign step number to order steps of processing */
  retain step 0;
  format inobs1 inobs2 outobs comma12.;
  /* process data steps with merge statements */
  if index(line,"MERGE ") then mergeflag=1;
  else if index(line,"BY ") then byflag=1;
  /* capture number of observations for data set */
  else if index(line,'NOTE: THERE WERE ') & index(line,' OBSERVATIONS READ FROM THE DATA
SET') then do;
    if mergeflag=1 then do;
      inobs1=input(scan(line,4,' '), 20.);
      infolder1=scan(line,11,' .');
      inds1=scan(line,12,' .');
      mergeflag=2;
    end; /* if mergeflag=1 */
    else if mergeflag=2 then do;
      inobs2=input(scan(line,4,' '), 20.);
      infolder2=scan(line,11,' .');
      inds2=scan(line,12,' .');
    end; /* else if mergeflag=2 */
    else do;
      inobs1=input(scan(line,4,' '), 20.);
      infolder1=scan(line,11,' .');
      inds1=scan(line,12,' .');
    end; /* else */
  end;
  else if index(line,'NOTE: THE DATA SET ') & index(line,' OBSERVATIONS AND ') &
index(line,' VARIABLES.') then do;
    outfolder=scan(line,5,' .');
    outds=scan(line,6,' .');
    outobs=input(scan(line,7,' '), 20.);
    outvars=input(scan(line,11,' .'), 20.);
  end;
  /* mark as data step or procedure */
  else if index(line, "REAL TIME ") > 0 then do;
    if index(line_lag1, 'DATA ') > 0 then
      stepname=scan(line_lag1, 2, ' ');
    else if index(line_lag1, 'PROCEDURE ') > 0 then
      stepname="PROC "||scan(line_lag1, 3, ' ');
    else
      stepname=scan(line_lag1, 3, ' ');
    if stepname="DATA" & mergeflag=2 then stepname="DATA/Merge";
    else if stepname="DATA" & inds1=" " then delete;
    /* mark beginning and end of program steps */
    else if stepname="INITIALIZATION" then stepname="*** Begin ***";
    else if stepname="INSTITUTE" then stepname="*** End ***";
    stepname=propcase(stepname);
    output;
    step + 1;
```

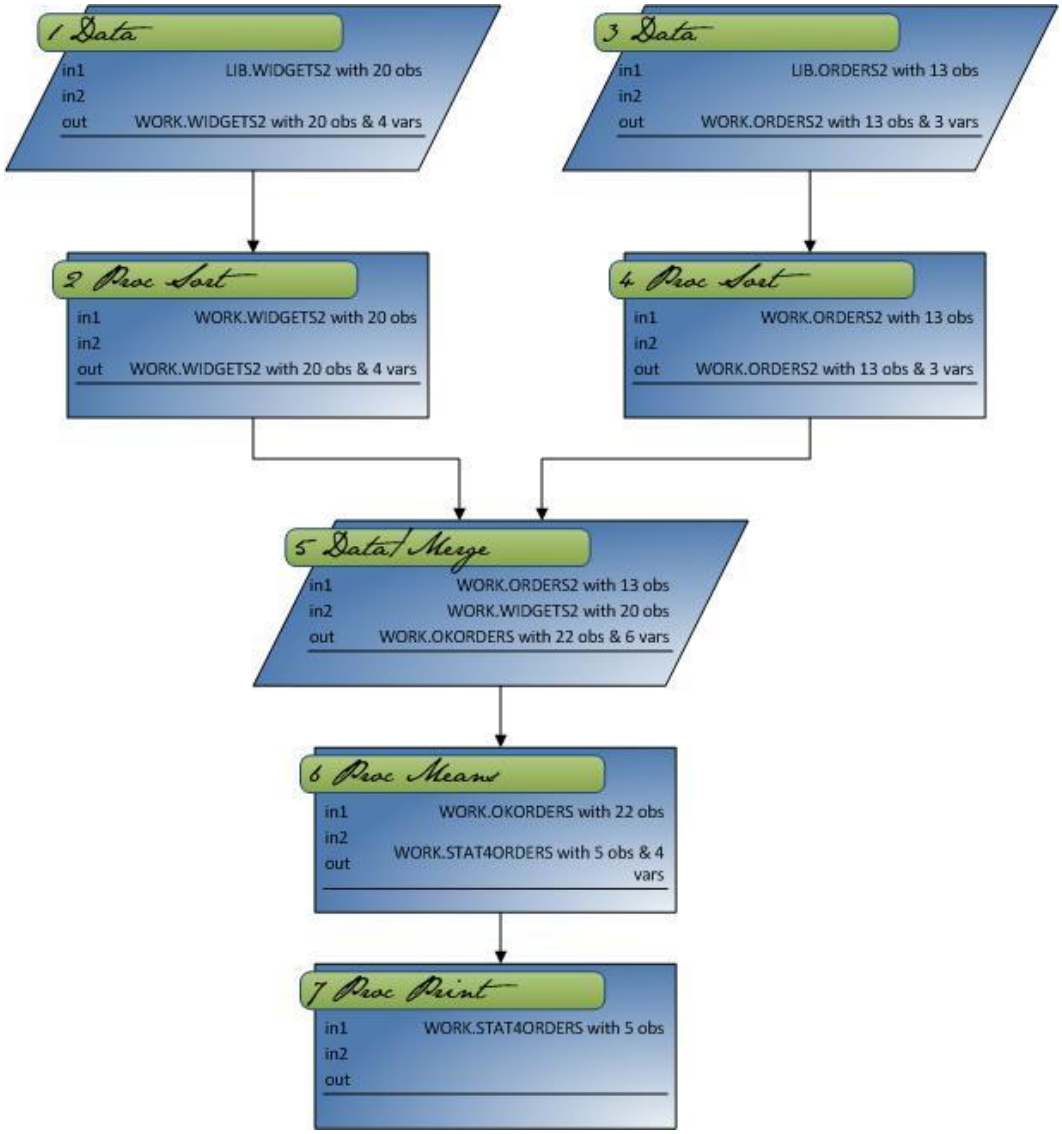
```
    array reset_n mergeflag byflag inobs1 inobs2 outobs outvars;
    array reset_c infolder1 infolder2 outfolder inds1 inds2 outds;
    do over reset_n;
      reset_n=.;
    end; /* do over reset_n */
    do over reset_c;
      reset_c=" ";
    end; /* do over reset_c */

  end; /*  if (index(line, "REAL TIME ") > 0) */
run;

/* combine fields for tabular output */
data parse4;
  length stepname in1 in2 out $80;
  set parse3;
  stepname = compress(step)||" "||stepname;
  in1=compress(infolder1)||"."||compress(inds1)||" with "||compress(inobs1)||" obs";
  if infolder2^=' ' then in2=compress(infolder2)||"."||compress(inds2)||" with
"||compress(inobs2)||" obs";
  if outfolder^=' ' then out=compress(outfolder)||"."||compress(outds)||" with
"||compress(outobs)||" obs & "||compress(outvars)||" vars";
  if infolder1^=" ";
  keep stepname in1 in2 out;
run;

/* export to spreadsheet */
proc export
  data=parse4
  outfile= "&lib.&log..xls"
  dbms=excel
  replace;
  sheet="Steps";
run;

%mend;
```

Table 1. Log Summary – Ouput to a Spreadsheet

stepname	in1	in2	out
1 Data	LIB.WIDGETS2 with 20 obs		WORK.WIDGETS2 with 20 obs & 4 vars
2 Proc Sort	WORK.WIDGETS2 with 20 obs		WORK.WIDGETS2 with 20 obs & 4 vars
3 Data	LIB.ORDERS2 with 13 obs		WORK.ORDERS2 with 13 obs & 3 vars
4 Proc Sort	WORK.ORDERS2 with 13 obs		WORK.ORDERS2 with 13 obs & 3 vars
5 Data/Merge	WORK.ORDERS2 with 13 obs	WORK.WIDGETS2 with 20 obs	WORK.OKORDERS with 22 obs & 6 vars
6 Proc Means	WORK.OKORDERS with 22 obs		WORK.STAT4ORDERS with 5 obs & 4 vars
7 Proc Print	WORK.STAT4ORDERS with 5 obs		

The spreadsheet contents can be used to create a flowchart. A flowchart is an excellent method to communicate with the “a picture is worth a thousand words” crowd. The flowchart below was created in Microsoft Visio 2007 using shapes created by the author for SAS diagrams. The Visio data can be manually entered or linked to an external source, such as the parsing macro’s output spreadsheet.

Figure 1. Flowchart of Program steps – Spreadsheet inputs to Visio



CONCLUSION

Oh! you are a great deal too apt, you know, to approve output in general.
You never see a fault in anything.
All the world are good and agreeable in your eyes.
I never heard you speak ill of a cross-tab in your life.

-misquoted from *Pride and Prejudice*

Quality Assurance in SAS programming tasks is essential. The review items for programmers are important for effective review in line with programming. The code samples presented here are tools for identifying and communication labeling issues in data sets. Succinctly documented in the output, the labeling issues can be resolved with the specification writer or subject matter experts. The parsed log code is a simple and powerful tool for communicating the processing of a program either with a tabular summary or a flowchart.

REFERENCES

Jane Austen, **Emma,** Oxford University Press, 1816.

Jane Austen, **Pride and Prejudice,** Oxford University Press, 1813.

Michael A. Raithel, **Programmatically Measure SAS® Application Performance On Any Computer Platform With the New LOGPARSE SAS Macro**, SUGI 30, 2005.

Rick Mitchell, **Finding Your Mistakes Before They Find You: A Quality Approach For SAS® Programmers**, SAS Global Forum 2007.

Ron Cody, **Cody's Data Cleaning Techniques Using SAS Software**, SAS Institute, 2006.

DISCLAIMER

The contents of this paper are the work of the author and do not necessarily represent the opinions, recommendations, or practices of Westat.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. You may write the author on white paper with a well-honed quill pen at the following address, or email deborahposner@westat.com.

Deborah Posner
Westat
1600 Research Boulevard
RW2533
Rockville, MD 20850

APPENDIX – DATA FOR EXAMPLES

WIDGETS DATA SET

WidgetID	Size	Color	InStock
101	XL	8	67
102	M	4	28
103	XL	5	22
104	M	5	
105	M	7	13
106	XL	7	48
107	S	8	25
108	L	8	7
109	XL	7	
110	XL	3	39

FORMATS FOR SIZE AND COLOR

```
PROC FORMAT;
  value $sizef /* format for Size */
    'XS'="Extra Small"
    'S'="Small"
    'M'="Medium"
    'XL'="Extra Large";
  value colorf /* format for Color */
    1="Red"
    2="Orange"
    3="Yellow"
    4="Green"
    5="Blue"
    6="Indigo"
    7="Violet";
run;
```