Abstract
Writing PROC REPORT calls is an effective method for making presentation-ready listings. Or is it? First, writing PROC REPORT calls requires knowledge of PROC REPORT. Second, a PROC REPORT call can be a lengthy ordeal with specifying the column information, followed by all the DEFINE statements. Thus, they do not lend themselves to be generic, reusable code. Third, variables with long lengths cannot be printed without a DEFINE statement, and PROC REPORT will split the column header within a word if the word is longer than the length of the column width.

Your assignment: create a generic data listing macro that delivers presentation-ready reports. It must have many of the capabilities of traditional PROC REPORT listings without requiring the user to know PROC REPORT. It must be generic enough to be used for most listings, and must address all of the above issues. Oh, and we need this macro in three days. A sample call:

```
%LIST(data=final, group=treatment sex, display=ptno startdt stopdt, skip=sex);
```

An impossible task? Not if you are SAS® knowledgeable and follow my 10 steps to a generic data listing macro. Although this paper will not give you all the code, it will provide the steps necessary to develop your own listing tool.

The 10 Steps to Generic Data Listings
“Generic report” is one of the constant buzzwords at SAS conferences. Every year, several programmers present the latest and greatest attempt at producing a generic report writer. Although their efforts are commendable, as a programmer I often find their presentations aloof. Either the presentation is a marketing demonstration, or the useful programming techniques are needles in an hour-long presentation haystack.

In my opinion, the best presentations at SAS conferences are those that show programming techniques in the real world. They need to teach how to use the programming techniques, not just show the outcome of the end result.

In this paper, I will discuss the 10 steps I took to create a generic data listing macro in three days:

1. Macro Name and Parameters
2. Prep Work – Initialize Settings and Variables
3. Change Requested Labels and Formats
4. Make Meta-Data of Input Data Set
5. Set Justification and Widths
6. Make Corresponding Data Sets for the Parameters
7. Combine Meta-data and Determine Proper Column Widths
8. Create Macro Variables For PROC REPORT
9. Build the PROC REPORT
10. Clean Up

I anticipate the reader will find several techniques and observations presented in this paper that could be used in writing other standard or generic macros.

Step 1: Macro Name and Parameters
Before writing any flow charts, program design, or code, you need to determine the name of the macro and the minimum input parameters. If you were lucky enough to receive user requirements (or unlucky in some cases), then these will already be determined or stated. In our case, we are simply asked to create general PROC REPORT listings. By general, I mean straightforward listings with no COMPUTE statements, statistics, or other advanced features.

First, what should we name the macro? My programming experience endorses the KISS method (Keep It Simple Stupid). If one cannot remember the macro name, then one is not going to use the macro. Since LIST is a reserved SAS word and cannot be used as a macro name, let’s use %LISTING because our macro produces listings.

Second, what parameters will our new macro contain? Begin by listing the useful features of PROC REPORT. I emphasize “useful” because PROC REPORT has many features. For general listings, these features can be reduced to a short list that most reports utilize:
- Input data set
- Group or order variables
- Display variables
- Specify formats
- Specify labels
- Specify widths
- Skip line or page after variable
- WHERE clause
- Specify spacing between columns
- Specify split character

There are other features that you can add to the list and subsequently to the macro. However, for this paper we will concentrate on the above list.

Now, we have a foundation to begin designing our macro. How do we want the user to specify all of the above? Remember the KISS philosophy: the more obscure the input parameter name, the less it will be used. Also, the more complicated the input parameter values, the less likely the parameter will be used. Therefore, I suggest making each bullet point its own parameter, using the same name as the function in PROC REPORT.

This gives us the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td>Input data set</td>
</tr>
<tr>
<td>GROUP</td>
<td>Variables to group/order</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>Variables to display</td>
</tr>
<tr>
<td>WHERE</td>
<td>WHERE clause</td>
</tr>
<tr>
<td>FORMAT</td>
<td>Change formats</td>
</tr>
<tr>
<td>LABEL</td>
<td>Change labels</td>
</tr>
<tr>
<td>WIDTH</td>
<td>Change widths</td>
</tr>
<tr>
<td>SKIP</td>
<td>Skip a line after variable(s)</td>
</tr>
<tr>
<td>PAGE</td>
<td>Skip a page after variable</td>
</tr>
<tr>
<td>SPACING</td>
<td>Spacing between columns</td>
</tr>
<tr>
<td>SPLIT</td>
<td>Split character</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td>final</td>
</tr>
<tr>
<td>GROUP</td>
<td>treatment sex</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>ptno startdt stopdt</td>
</tr>
<tr>
<td>WHERE</td>
<td>ptno &gt; 0</td>
</tr>
<tr>
<td>FORMAT</td>
<td>sex sex1s.</td>
</tr>
<tr>
<td>LABEL</td>
<td>sex=&quot;Gender&quot;</td>
</tr>
<tr>
<td>WIDTH</td>
<td>startdt=9 stopdt=9</td>
</tr>
<tr>
<td>SKIP</td>
<td>sex</td>
</tr>
<tr>
<td>PAGE</td>
<td>treatment</td>
</tr>
<tr>
<td>SPACING</td>
<td>2</td>
</tr>
<tr>
<td>SPLIT</td>
<td>$</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Debug</td>
</tr>
<tr>
<td>MAXWIDTH</td>
<td>20</td>
</tr>
</tbody>
</table>

Finally, in creating the generic data listing macro, we want to be sensitive to other SAS programs and the users. By this, I mean that we want to restrict the user as little as possible, as well as not pollute the SAS environment. Therefore, any data sets, variables, or macro variables that we create will follow a naming convention that will be specific to our macro: all will start with ‘X’. Second, we will delete all data sets created by the macro, because nobody likes other people’s trash.

Step 2: Prep Work

All macros should have prep work where they initialize settings and setup the environment. Although this may seem arduous or tedious, it is good practice and will save lots of programming time later in the macro.

First, we do not want to pollute or tarnish the global macro variable environment, so be sure to use %LOCAL to list as many of the local macro variables as you can that will be created. Just add them to the list as your code progresses.

Second, we do not want to fill the log with excessive information with messages from SYMBOLGEN and MLOGIC (unless you are required). So, our macro will automatically turn off these and similar log producing options. Instead, we will write to the log using %PUT statements the various steps the macro is taking. However, we do want to return the options to their original settings once the macro finishes. Therefore, be sure to capture the initial option settings into a macro variable before changing them. At the end of the program, reset these options to their original settings using this macro variable.
Also, we do not want our parameters to be case sensitive. There is no reason to require all the input to be uppercase or lowercase. Therefore, at the beginning of our macro we will uppercase all our non-numeric parameters except for the WHERE, LABEL, and SPLIT parameters as the user may desire using lowercase letters. Performing this function at the beginning allows us to omit using %UPCASE every time we check the value of a parameter later in the macro.

This is also a good place to start error checking. Nothing irritates users more than having a macro generate SAS errors with no explanation. A programmer should check for invalid input throughout the macro. If invalid input is detected, then the macro should print an explanation to the log and shut down in an orderly fashion. Here are some good things to check:

- Does the input data set exist?
- Are there common syntax errors such as having FORMAT contain an equal sign?
- Are the values of SPACING and MAXWIDTH numeric?
- Are GROUP and DISPLAY both null?

For the last bullet, one must decide whether this is a critical error or an allowable entry. In our macro, if both GROUP and DISPLAY are null, then the macro should list all the variables in the position they appear in the data set by treating them as grouped variables. So, be sure to check for this condition. If it is true, then set the parameter GROUP equal to the names of the variables in the order they appear in the input data set.

Finally, you will find it easier later on if you now quote the values contained in GROUP and DISPLAY by using the QUOTE function. Having the individual values quoted allows one to use the parameters in WHERE statements later in the macro. It also allows an opportunity to count the number of variables within each parameter. For those unfamiliar with the QUOTE function, the following code is quoting each variable found in the GROUP parameter while creating a new macro variable XGVAR. Thus, if GROUP=TREAT SEX, then XGVAR="TREAT" "SEX":

```sas
%let xgvar=; 
%let x=1; 
%do %while(%scan(&group,&x) ne and &x<50); 
 %let xgvar=&xgvar %sysfunc(quote(%trim(%scan(&group,&x)))); 
 %let x=%eval(&x+1); 
%end; 
```

Remember to keep track of how many GROUP and DISPLAY variables you have. Also create a macro variable for the total number of variables.

### Step 3: Change Requested Labels and Formats

There are many possible methods for having the user request changes to formats and labels. However, if we follow the KISS method, then the valid syntax for the LABEL and FORMAT parameters would match the data step syntax. Thus, if the user requests any changes to the labels or formats, we can simply create a new data set from the old data set using a data step and apply the label and format changes. Do not use a PROC DATASTETS statement to perform these changes as that would change the input data set:

```sas
** Check if LABEL or FORMAT requested; 
%if %length(|&label) > 1 or %length(|&format) > 1 %then 
do; 
data xdatal ; 
 set &data ; 

** Change labels if requested; 
%if %length(|&label) > 1 %then 
 %do; 
label &label; 
%end; 

** Change formats if requested; 
%if %length(|&format) > 1 %then 
 %do; 
 format &format; 
%end; 
r
run; 
%let data=xdatal; 
%end; 
```

### Step 4: Make Meta-Data of Input Data Set

This step is quick and easy. Perform a PROC CONTENTS with the OUT and NOPRINT option. However, since SAS version 7 and later versions no longer store variable names as solely uppercase, be sure to add a data step to set the values of NAME to...
uppercase. This will prevent possible issues later. By the way, throughout the macro I use PROC CONTENTS instead of PROC SQL and DICTIONARY.COLUMNS because I obtain the same information and avoid having to determine the LIBNAME value of the input data set.

Step 5: Set Justification and Widths
This step can be tricky for several reasons. To determine the proper width of a variable, it is important to remember that one must obtain the width associated with any applicable format, not just the length of the variable itself. For example, the length of the variable SEX may be $1 with values “M” or “F”. We do not want to set the width for SEX to 1. Instead, we want to set the width to the length of the format, which would be 6 to allow “Female”.

First, determine whether there are any user-defined formats for the requested variables in GROUP and DISPLAY. One accomplishes this by querying the meta-data where FORMAT does not have the values $ or null. Then, use PROC FORMAT to create a temporary data set containing the format names and lengths for the variables. Then, merge the length of the format onto the meta-data by giving it a new variable name. For this paper, we will call the new variable XFMTLEN because it is the format length.

Also, for aesthetics it is good to left justify character variables and right justify numeric variables. However, if a numeric variable has a format containing text (e.g. 1=”Male”, 2=”Female”), then the variable should be left justified. So, within a data step we will need to adjust the contents of the meta-data by setting the default width and justification for each requested variable.

Also, due to the nature of SAS and PROC REPORT, numeric variables with no assigned formats are assigned a BEST# format in PROC REPORT. This can cause display issues when used with the FLOW option. Thus, we must also adjust the formats for those variables without assigned formats.

As I mentioned, this step has some tricky parts due to how SAS stores information in data sets from PROC CONTENTS. Be sure to follow this logic:

1. Character variables should be left justified.
2. If the format length of a character variable is longer than the length of the variable, then set the width equal to the format length.
3. Character variables with no assigned format should use the value of LENGTH from PROC CONTENTS for the width. Also, populate a value for FORMAT to be used in PROC REPORT by setting it equal to “$|LENGTH.
4. If the character variable does have a format, then one will need to add the decimal to the end of the value since SAS does not store this in PROC CONTENTS.
5. Numeric variables with no user-defined format should be right justified.
6. Numeric variables with a user-defined format (FORMAT value not equal null) should be left justified. Set the width equal to the format length. Also add the decimal to the end of the FORMAT value.
7. Numeric variables with no format assigned need to have the length of the variable assigned as its format.

I have provided the code since the logic may be difficult to follow:

```sas
** xfmtlen = length of format;
** xinfoz = meta-data data set;
** &xgvar = GROUP variables;
** &xdvar = DISPLAY variables;

data xinfo;
  length format $30 justify $10;
  set xinfoz ;
  where upcase(name) in (&xgvar &xdvar);
  **correct character variables;
  if type=2 then do;
    justify='left';
    if xfmtlen>length then
      length=xfmtlen;
    if format in('$$' '') then
      format=compress('$$'||length||'.');
    else format=compress(format||'.');
  end;
  **correct numeric variables;
  else if type=1 then do;
    if format eq '' then
      justify='right';
    else justify='left';
```

Pharmaceuticals NESUG 15
if xfmtlen>0 then
   do;
      format=compress(format||'.');
      length=xfmtlen;
   end;
   if format='' then
      format=compress(length||'.'||formatd);
end;
run;

Below is a sample data set if the user submitted the following macro call:

%listing(data=test, group=country sex, display=ptno, width=sex=8);

XINFO (Meta-data)

<table>
<thead>
<tr>
<th>NAME</th>
<th>FORMA</th>
<th>JUSTIFY</th>
<th>LENGTH</th>
<th>XFMTELEN</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNTRY</td>
<td>$8.</td>
<td>left</td>
<td>8</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>PTNO</td>
<td>8.0</td>
<td>right</td>
<td>8</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>SEX</td>
<td>$SEX.</td>
<td>left</td>
<td>6</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

Step 6: Make Corresponding Data Sets for the Parameters

There are various techniques for accomplishing steps 6 and 7. The ultimate goal is to obtain macro variables containing the appropriate options for PROC REPORT so that one can loop through the DEFINE statements and assign attributes by using macro variables. The method I have chosen is to create individual data sets for each necessary parameter (GROUP, DISPLAY, and WIDTH). After creating the data sets, I will then merge the data sets together, creating meta-data for each variable. I can then create corresponding macro variables that will provide the correct values within the DEFINE statement loops.

Each data set will contain the same information: the NAME of the variable, and the SORT order (i.e. column order). The routine will be the same for each parameter, so create a loop or a nested macro to do the following tasks:

1. Scan for a value
2. Set NAME equal to the value
3. Set SORT equal to the looping number.
4. Sort the data set by NAME to ensure proper merge.

If the parameter value is null, then create an empty data set. This will prevent an error when merging the data later. Below are some sample data sets if the user submitted the above macro call:

XGRP (Group Parameter)

<table>
<thead>
<tr>
<th>NAME</th>
<th>DIS</th>
<th>XSORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNTRY</td>
<td>group</td>
<td>1</td>
</tr>
<tr>
<td>SEX</td>
<td>group</td>
<td>2</td>
</tr>
</tbody>
</table>

XDIS (Display Parameter)

<table>
<thead>
<tr>
<th>NAME</th>
<th>DIS</th>
<th>XSORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTNO</td>
<td>display</td>
<td>1</td>
</tr>
</tbody>
</table>

XWID (Width Parameter)

<table>
<thead>
<tr>
<th>NAME</th>
<th>WIDTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEX</td>
<td>8</td>
</tr>
</tbody>
</table>

Here is the sample code for this logic:

%macro xmac2(var=,svar=);
  %if %length(|&&&var) > 1 %then
    %do;
      %let x=1;
      %do %while (%scan(&&&var,&x) ne and &x <50);** Get variable name;
        %let xx&svar.&x=%scan(&&&var,&x);
        %let x=%eval(&x+1);
      %end;
      %let xx&svar.cnt=%eval(&x-1);
      data x&svar;
        length &svar $8 name $32 ;
        %do i=1 %to &xx&svar.cnt;
          name=upcase("&&&xx&svar.&i");
          &svar="&var";
          xsort=&i;
          output;
        %end;
        run;
    %end;
  %else
    %do;
      data x&svar;
      length name $8;
      stop;
      run;
  %end;
%end;
%end;
%mend xmac2;

%mac2(var=group,svar=grp); **GROUP;
%mac2(var=display,svar=dis);**DISPLAY;

For the WIDTH parameter that contains an equal sign ‘=’ and corresponding value, the process is similar, but one will need to scan for the variable name and value.

Now that the parameter values are in data sets, it is a good time to perform additional error checking. One such check is whether there are duplicate variable names in GROUP and DISPLAY. Another good check is whether the variables requested in GROUP and DISPLAY are part of the input data set.

**Step 7: Combine Meta-data and Determine Proper Column Widths**

Merge the meta-data created earlier with the three new data sets by NAME. This will create new meta-data containing almost all the information we need for the PROC REPORT (partial display):

<table>
<thead>
<tr>
<th>NAME</th>
<th>FORMA T</th>
<th>GRP</th>
<th>DIS</th>
<th>XSORT</th>
<th>WIDTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNTRY</td>
<td>$8.</td>
<td>group</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>PTNO</td>
<td>8.0</td>
<td>display</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>SEX</td>
<td>$SEX.</td>
<td>group</td>
<td></td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

However, there is still one more task: verifying the proper width of the variable. If the variable width is set too small, then PROC REPORT will split the column header within a word. If the variable width is set too long, then PROC REPORT will not print the table. Thus, we need to find the length of the longest word in the column header. Then, we can make sure the value of the width is between this length and the value of MAXWIDTH. The following code performs this task:

```
------------------------------------------;
**Determine length of longest word;
------------------------------------------;
xcount=0;
xlen1=0;
do until(xword = ' ');
xcount=xcount+1;
xword=scan(label, xcount, ' ');
xlen=length(xword);
xlen1=max(xlen,xlen1);
end;
```

```
***Determine width by taking the maximum
**between the length of the longest **word
and the minimum value between **the maximum
column width and **default column width;
***                          
xwidth=max(xlen1,
    (min(&maxwidth,length)));
```

**Final Meta-data Partial Display**

<table>
<thead>
<tr>
<th>NAME</th>
<th>FORMA T</th>
<th>JUSTIF</th>
<th>XWIDTH</th>
<th>XSORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNTRY</td>
<td>$8.</td>
<td>left</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>PTNO</td>
<td>8.0</td>
<td>right</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>SEX</td>
<td>$SEX.</td>
<td>left</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

**Step 8: Create Macro Variables For PROC REPORT**

Now, we are ready to make our macro variables that can be used in the DEFINE statements. To accomplish this task, I use PROC SQL to create the macro variables for the attributes that correspond to each variable in the order that they will appear on the report:

```
proc sql noprint;
select trim(name), xwidth,
    trim(justify), trim(format)
into :xnam1-:xnam&xnvar,
    :xwid1-:xwid&xnvar,
    :xjus1-:xjus&xnvar,
    :xfmt1-:xfmt&xnvar
from xinfo1
order by grp,dis,xsort;
quit;
```

**Macro Variables**

<table>
<thead>
<tr>
<th>Suffix</th>
<th>XNAM</th>
<th>XWID</th>
<th>XJUS</th>
<th>XFMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>COUNTRY</td>
<td>14</td>
<td>left</td>
<td>$8.</td>
</tr>
<tr>
<td>2</td>
<td>SEX</td>
<td>8</td>
<td>left</td>
<td>$SEX.</td>
</tr>
<tr>
<td>3</td>
<td>PTNO</td>
<td>8</td>
<td>right</td>
<td>8.0</td>
</tr>
</tbody>
</table>

**Step 9: Build the PROC REPORT**

Finally, you are ready to create your PROC REPORT. State your beginning PROC REPORT line using the parameter SPACING to set the spacing, and SPLIT for the split character definition. Add the WHERE parameter. The COLUMN statement is simply the parameters GROUP and DISPLAY. Now comes the DEFINE statements.
Since you kept track of the number of GROUP variables, use the macro variable in a %DO loop to set the DEFINE statements for the GROUP variables:

```sas
%do i=1 %to &xgvarn;
   DEFINE &&xnam&i / ORDER
       ORDER=INTERNAL ID FORMAT=&&xfmt&i
       WIDTH=&&xwid&i &&xjus&i FLOW;
%end;
```

For the DISPLAY variables, again use a %DO loop, starting with the number of GROUP variables +1, and ending with number of GROUP and DISPLAY variables:

```sas
%do i=%eval(&xgvarn+1) %to &xnvar;
   DEFINE &&xnam&i / DISPLAY
       ORDER=INTERNAL FORMAT=&&xfmt&i
       WIDTH=&&xwid&i &&xjus&i FLOW ;
%end;
```

For the SKIP parameter, scan off each value to create an individual line:

```sas
%let x=1;
%if %length(|&SKIP) > 1 %then
   %do;
   %do %while(%scan(&SKIP,&x) ne and &x<10);
       BREAK AFTER %scan(&SKIP,&x) / SKIP;
   %let x=%eval(&x+1);
   %end;
%end;
```

Create a similar block of code for PAGE, but use BREAK AFTER / PAGE instead.

**Step 10: Clean Up**
To clean up, simply delete all the temporary data sets that were created. Since our naming convention created all our data sets starting with ‘X’, we can use the following code:

```sas
 proc datasets lib=work mt=data nolist;
   delete x: ;
 quit;
```

Also, be sure to reset the options to their original settings.

**Conclusion**
This paper resulted from a personal challenge I made to myself that creating a generic listing macro is neither difficult nor time consuming. The task needs a well-designed plan while keeping in mind the user and the abilities of PROC REPORT. Most important, one needs to “keep it simple stupid” regarding the parameters. The method outlined in this paper allows for easy enhancements to this macro such as letting the user specify the justification of variables or variables to not print. But do not make your generic data listing macro too complicated.

Although the generic data listing macro is not able to perform all the functions and features of PROC REPORT, it is not designed to replace PROC REPORT. A tool that can perform 75% of the work and is used by 90% of the employees is far more resource efficient than one that does 99% of the work but is used by only 30% of the employees.

**Acknowledgments**
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 registered trademarks or trademarks of their respective companies.

**Contact Information**
The author welcomes comments, suggestions, or questions:

Chris Moriak  
Quantitative Decision Sciences  
AstraZeneca  
1800 Concord Pike  
PO Box 15437  
Wilmington, DE 19850  
Tel: 302-885-9977  
E-mail: christopher.moriak@astrazeneca.com
Appendix A: Data and Output Examples

Code

*** Sample Data ;
proc format ;
  value race
    1="Caucasian Decent"
    2="Afro-American Decent"
    3="Other";

  value $sex
    "M"="Male"
    "F"="Female"
  ;
run;

data test ;
   attrib study length=$25 format=$25. label="Study";
   attrib ptno length=8 label="Patient Number";
   attrib country length=$8 label="Investigator's Country";
   attrib age length=8 label="Age";
   attrib sex length=$1 format=$sex. label="Sex";
   attrib race format=race. label="Race";
   input study $ ptno country $ age sex $ race ;
   cards;
   105.122 101 US 24 M 1
   105.122 102 US 26 F 2
   105.122 103 US 27 M 2
   105.122 104 US 23 M 1
   105.122 105 US 22 M 1
   105.122 106 US 25 F 3
   105.122 107 US 26 M 1
   105.234 101 US 28 M 1
   105.234 102 US 25 F 2
   105.234 103 US 23 F 3
   105.234 104 US 22 F 1
   105.234 105 US 21 M 1
  ;
run;
### PROC REPORT Default Output

```plaintext
proc report data=test nowd;
run;
```

<table>
<thead>
<tr>
<th>Study</th>
<th>Patient Number</th>
<th>Country</th>
<th>Age</th>
<th>Sex</th>
<th>Race</th>
</tr>
</thead>
<tbody>
<tr>
<td>105.122</td>
<td>101</td>
<td>US</td>
<td>24</td>
<td>Male</td>
<td>Caucasian Decent</td>
</tr>
<tr>
<td>105.122</td>
<td>102</td>
<td>US</td>
<td>26</td>
<td>Female</td>
<td>Afro-American Decent</td>
</tr>
<tr>
<td>105.122</td>
<td>103</td>
<td>US</td>
<td>27</td>
<td>Male</td>
<td>Afro-American Decent</td>
</tr>
<tr>
<td>105.122</td>
<td>104</td>
<td>US</td>
<td>23</td>
<td>Male</td>
<td>Caucasian Decent</td>
</tr>
<tr>
<td>105.122</td>
<td>105</td>
<td>US</td>
<td>22</td>
<td>Male</td>
<td>Caucasian Decent</td>
</tr>
<tr>
<td>105.122</td>
<td>106</td>
<td>US</td>
<td>25</td>
<td>Female</td>
<td>Other</td>
</tr>
<tr>
<td>105.122</td>
<td>107</td>
<td>US</td>
<td>26</td>
<td>Male</td>
<td>Caucasian Decent</td>
</tr>
<tr>
<td>105.234</td>
<td>101</td>
<td>US</td>
<td>28</td>
<td>Male</td>
<td>Caucasian Decent</td>
</tr>
<tr>
<td>105.234</td>
<td>102</td>
<td>US</td>
<td>25</td>
<td>Female</td>
<td>Afro-American Decent</td>
</tr>
<tr>
<td>105.234</td>
<td>103</td>
<td>US</td>
<td>23</td>
<td>Female</td>
<td>Other</td>
</tr>
<tr>
<td>105.234</td>
<td>104</td>
<td>US</td>
<td>22</td>
<td>Female</td>
<td>Caucasian Decent</td>
</tr>
<tr>
<td>105.234</td>
<td>105</td>
<td>US</td>
<td>21</td>
<td>Male</td>
<td>Caucasian Decent</td>
</tr>
</tbody>
</table>

### Macro Call #1: %LISTING Default Output

```plaintext
%listing(data=test);
```

<table>
<thead>
<tr>
<th>Study</th>
<th>Patient Number</th>
<th>Country</th>
<th>Age</th>
<th>Sex</th>
<th>Race</th>
</tr>
</thead>
<tbody>
<tr>
<td>105.122</td>
<td>101</td>
<td>US</td>
<td>24</td>
<td>Male</td>
<td>Caucasian Decent</td>
</tr>
<tr>
<td>105.122</td>
<td>102</td>
<td>US</td>
<td>26</td>
<td>Female</td>
<td>Afro-American Decent</td>
</tr>
<tr>
<td>105.122</td>
<td>103</td>
<td>US</td>
<td>27</td>
<td>Male</td>
<td>Afro-American Decent</td>
</tr>
<tr>
<td>105.122</td>
<td>104</td>
<td>US</td>
<td>23</td>
<td>Male</td>
<td>Caucasian Decent</td>
</tr>
<tr>
<td>105.122</td>
<td>105</td>
<td>US</td>
<td>22</td>
<td>Male</td>
<td>Caucasian Decent</td>
</tr>
<tr>
<td>105.122</td>
<td>106</td>
<td>US</td>
<td>25</td>
<td>Female</td>
<td>Other</td>
</tr>
<tr>
<td>105.122</td>
<td>107</td>
<td>US</td>
<td>26</td>
<td>Male</td>
<td>Caucasian Decent</td>
</tr>
<tr>
<td>105.234</td>
<td>101</td>
<td>US</td>
<td>28</td>
<td>Male</td>
<td>Caucasian Decent</td>
</tr>
<tr>
<td>105.234</td>
<td>102</td>
<td>US</td>
<td>25</td>
<td>Female</td>
<td>Afro-American Decent</td>
</tr>
<tr>
<td>105.234</td>
<td>103</td>
<td>US</td>
<td>23</td>
<td>Female</td>
<td>Other</td>
</tr>
<tr>
<td>105.234</td>
<td>104</td>
<td>US</td>
<td>22</td>
<td>Female</td>
<td>Caucasian Decent</td>
</tr>
<tr>
<td>105.234</td>
<td>105</td>
<td>US</td>
<td>21</td>
<td>Male</td>
<td>Caucasian Decent</td>
</tr>
</tbody>
</table>
Macro Call #2: Enhanced %LISTING Output

%listing(data = test,
  group   = study country sex,
  display = ptno race,
  label   = ptno="Subject",
  width   = study=10,
  skip    = sex,
  where   = study="105.122" );

<table>
<thead>
<tr>
<th>Study</th>
<th>Country</th>
<th>Sex</th>
<th>Subject</th>
<th>Race</th>
</tr>
</thead>
<tbody>
<tr>
<td>105.122</td>
<td>US</td>
<td>Female</td>
<td>102</td>
<td>Afro-American Decent</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>106</td>
<td>Other</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Male</td>
<td>101</td>
<td>Caucasian Decent</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>103</td>
<td>Afro-American Decent</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>104</td>
<td>Caucasian Decent</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>105</td>
<td>Caucasian Decent</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>106</td>
<td>Caucasian Decent</td>
</tr>
</tbody>
</table>