Why and How to Use SAS® Macro Language: Easy Ways to Get More Value and Power from Your SAS® Software Tools

LeRoy Bessler PhD, Bessler Consulting and Research, Strong Smart Systems™
Mequon, Wisconsin, USA, Le_Roy_Bessler@wi.rr.com

ABSTRACT

SAS Macro Language and macro variables (a.k.a. symbolic variables) let your SAS program do things for you that would ordinarily require you to make manual changes to your code every time you run it (e.g., changing title text, changing data filters, graph controls, etc.)

Whenever you change your code, not only does it take extra time, but also there is a risk of inadvertently introducing an error.

SAS Macro Language makes code situation-adaptive. It can adapt to changes in data, run date, data date, data date range, etc. It is an essential tool for Software-Intelligent Application Development, which delivers Reliability, Reusability, Maintainability, Extendability, and Flexibility.

Strong Smart Systems are essential for production programs that are run by automatic batch scheduling or must respond in a custom fashion in real-time to interactive requests. Such programs must run hands-off every time.

Code using macro language also saves time and reduces error risk for ad hoc work. Any supposedly one-time ad hoc code you build has the possibility of being needed again, very likely with minor variation, to satisfy a future request for analysis, report, or data.

Programs that are basically similar, but with minor variations in purpose and output, need to differ only in macro invocation code, which is easy to find and understand, as compared to changes buried somewhere in a huge number of lines of SAS code.

This introductory tutorial will get you started using a powerful feature of Base SAS software that you already have, and which can be used with all of your other SAS software products as well.

INTRODUCTION

While working on the first edition of this paper, I encountered a question posted at the SAS Professional Forum LinkedIn group: “What is Macros simply?” A long discussion then ensued about whether this group was the place to post such a basic and broad question for which the answer is well documented elsewhere. No one really answered the question directly. Finally, someone said: “Isn't the simple answer as follows: macros are used to make tasks using the application less repetitive. They are a tool that allows a programmer to enable code reuse."

At that point, since one of my obsessions is completeness, I had to say: “Enablement of code reusability is only part of the answer. SAS macro language and SAS macro variables are two of the essential keys to what I call Software-Intelligent Application Development, which delivers Reliability, Reusability, Maintainability, Extendibility, and Flexibility. You can create code that adapts to changes in data, run dates, data dates, data date ranges, filter requirements, etc., eliminating any need for manual changes to the code. If code runs automatically in computer-scheduled batch or is invoked by live users in real-time, it has to do the right thing the first time every time without hands-in modification, and the environment is changing. Even if you want to use the SAS macro facility only for reusability, I contend that SAS macro language is the most important second language for any professional SAS programmer. But I admit to SAS bias: “If you cannot do it with SAS software, you probably don’t really need to do it.”

Here, for you, I confess that I did not step up to macro language readily. I had been using SAS software for almost ten years and abstaining from investigating how to use SAS Macro Language. I would see use of macros in other people’s code (or in software that was built with SAS macros), but had found all of those percent signs and ampersands to be weird and off-putting, and I did not need to code or use macros to do my job. Then in 1987 I was working on a project where I wanted to do something with SAS/GRAPH®, which I had already mastered over a period of seven years, but the only way to do it, I was told by SAS Technical Support, was to use macro language.

SAS Macro Language enabled me to build my first Software-Intelligent production batch application. I needed my code to automatically do things to graph axes, titles, subtitles, data selection, etc. without any manual intervention by

me every time that it ran. I had very specific design and function objectives, and native Base SAS and native SAS/GRAPH would not meet those objectives without customization every time that the code would run. This automation replaced a manual-intensive effort by several other people, using SAS and other tools, and their process had been error vulnerable and the outputs had been inelegant. I eventually published a paper for SAS users about what I did (see Reference 1). The experience made me an advocate for the power of SAS Macro Language.

I am convinced that it is the most important second language for any serious SAS programmer. I will do my best to help get you started in what follows. Also, I have put together a bibliography of other resources you might find useful.

If you study SAS Macro Language in a formal manner or read any of the books about it, you are likely to encounter, as an entry point, a discussion of the internals of how SAS deals with macros. It gets into the tokenizer, a stack, and other abstruse concerns that you really do not need to know about, much less try to understand, in order to be an effective and safe user of SAS macros. I have almost thirty years of experience and success with the SAS macro facility without mastering this. Smarter people than I might disagree with this approach. Once you are productive with SAS Macro Language, and have time that you cannot put to better use, you might decide to investigate the macro language facility infrastructure. However, there are some subtleties worth being aware of. See, e.g., this tip: http://blogs.sas.com/content/publishing/2015/04/01/sas-authors-tip-macro-language-timing-is-everything/

Here is the plan for the tutorial:

1. WHERE TO ACCESS OR STORE SAS MACROS
2. WHAT IS A MACRO AND HOW DO I USE ONE? Two super simple examples
3. MACRO VARIABLES (a.k.a., SYMBOLIC VARIABLES): The simplest macro language tools & how to use them
4. MACRO LANGUAGE STATEMENTS THAT YOU CAN USE ANYWHERE
5. MACRO FUNCTIONS THAT YOU CAN USE ANYWHERE
6. SCOPE OF MACRO VARIABLES: Global Versus Local
7. OTHER WAYS TO ASSIGN VALUES TO MACRO VARIABLES: DATA Step and PROC SQL
8. WHAT TO DO WHEN THE MACRO FUNCTION YOU NEED DOES NOT EXIST
9. MACRO FUNCTIONS FOR ARITHMETIC
10. CONDITIONAL PROCESSING WITH MACRO LANGUAGE
11. FULL AUTOMATION TO ELIMINATE ANY NEED FOR PROGRAMMER INTERVENTION
12. COMMON PROCESSING FOR A LIST OF NUMBER-SUFFIXED MACRO VARIABLES
13. A SIMPLIFIED VERSION OF ONE OF MY FAVORITE APPLICATIONS OF SAS MACRO LANGUAGE

I am grateful to Laura MacBride for her helpful review of the 2012 edition of this paper, but any lack of clarity or other imperfections are my responsibility.

WHERE TO ACCESS OR STORE SAS MACROS

Some macros are shipped with SAS software. To use them you do nothing special, except invoke them (with any needed parameters—more about parameters later) like this, anywhere appropriate in your code:

```
%SomeMacroShippedWithSAS; /* This macro would be one with no parameters,
    or is being invoked using the defaults assigned in the definition of the macro. */
```

If you create your own macros, and you do not want to manually paste them into your code every time, you can store them anywhere that SAS has READ access: on your hard drive or some location on your enterprise LAN.

To use one of your own macros in your code that is remotely stored as above, you have two choices. Before invocation of the macro with **`%YourMacroName;`**
your code must contain either statement: **`%INCLUDE "PathToYourFolderOfMacros\YourMacroName.sas";`**
or, as a better method, statement: **`OPTIONS SASAUTOS=("PathToYourFolderOfMacros" SASAUTOS);`**

The parenthesis above can contain any number of macro libraries, but must always contain SASAUTOS. When SAS starts, SASAUTOS points to the default location where macros shipped with SAS are stored. The above OPTIONS statement tells SAS to look for macros first in PathToYourFolderOfMacros. If you name your macro with a macro name already used by SAS, your version will be used. It's not a good idea to use SAS's own macro names. For any number of macro libraries, they are searched in list order from left to right. In this paper, the OPTIONS statement will NOT be explicitly used, but the macro library concept assures that a macro needs to be written only once, and that any program with access to that library storage location will be able to use the macro.

WHAT IS A MACRO AND HOW DO I USE ONE? Two super simple examples

It is a way to create dynamic code that gets finalized at run time based on parameters that are passed to it.

This introduction will show you:

code that is submitted;
what you can see in the SAS log IF you ask SAS to show you the code that is actually built and run for you;
and
what you can see in the SAS log IF you DO NOT ask SAS to show you that code as resolved and run.

Here is a simple example of a non-trivial macro:

```
%macro MyFirstMacro(data=);

options nocenter nodate nonumber;
proc print data=&data; /* &data is to be replaced by value assigned with data= */
run;

%mend  MyFirstMacro;
```

A trivial macro would be one where the name of the data set to be printed was already hard-coded inside the macro, instead of being a macro invocation parameter. The non-trivial macro can be used to print ANY SAS data set, with NO manual modification of the code internal to the macro.

The macro can be stored in a folder on your hard disk or anywhere on the LAN where you have READ access, as explained in the prior section.

If you are not accessing a macro from elsewhere, then it must precede its first invocation in your SAS program.

Here is how you can use the macro shown above (presuming that you have already submitted the macro definition code above or are accessing the macro from a macro library):

```
options MPRINT; /* to see the code after replacement and what is actually run */
%MyFirstMacro(data=sashelp.class);
```

Here is what you can find in the SAS log (but with time results specific to your computer):

```
1    %macro MyFirstMacro(data=);
2
3    options nocenter nodate nonumber;
4    proc print data=&data; /* &data is to be replaced by value assigned with data= */
5    run;
6
7    %mend  MyFirstMacro;
8
9    options MPRINT; /* to see the code after replacement and what is actually run */
10   %MyFirstMacro(data=sashelp.class);
MPRINT(MYFIRSTMACRO):   options nocenter nodate nonumber;
MPRINT(MYFIRSTMACRO):   proc print data=sashelp.class;
MPRINT(MYFIRSTMACRO):   run;

NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.06 seconds
      cpu time            0.06 seconds
```

Here is an excerpt of the top of the result:

```
The SAS System

Obs    Name       Sex    Age    Height    Weight

  1    Alfred      M      14     69.0      112.5
  2    Alice       F      13     56.5       84.0
  3    Barbara     F      13     65.3       98.0
```

```
  4    Carol       F      14     62.8      102.5

< excerpt cut off here by author >
```

The MPRINT option is off by default. If you were in the same SAS session as that used to run the code above, you can turn off MPRINT and rerun the code as follows:

```
%macro MyFirstMacro(data=);

options nocenter nodate nonumber;
proc print data=&data; /* &data is to be replaced by value assigned with data= */
run;

%mend  MyFirstMacro;

options NOMPRINT; /* this is actually the default */
%MyFirstMacro(data=sashelp.class);
```

Then you will find this in your SAS log:

```
1    %macro MyFirstMacro(data=);
2
3    options nocenter nodate nonumber;
4    proc print data=&data; /* &data is to be replaced by value assigned with data= */
5    run;
6
7    %mend  MyFirstMacro;
8
9    options NOMPRINT; /* this is actually the default */
10   %MyFirstMacro(data=sashelp.class);

NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

I.e., you do not see how SAS actually converted your macro invocation into the code that it ran for you.

Tip: Besides the MPRINT option there is also an MPRINTNEST option which affects how resolved code is flagged when a macro is invoked inside of another macro. Also, you can use the MFILE option to get a copy of MPRINT SAS log content in an external file. See the Online Documentation for details.

You can make the functions that a macro performs as complicated as you want, write the code for them only one time, and provide as many control parameters as you like. Then you do not need a separate complicated (or simple) program for each function. Use only one macro, and get it to do what you want by setting the parameter values.

Here is a slightly fancier example:

```
%macro MySecondMacro(data=,HowMany=);

options nocenter nodate nonumber;
options obs=&HowMany;
title "Listing of First &HowMany observations in &data";
proc print data=&data;
run;
options obs=max;

%mend MySecondMacro;
```

Here is how you could use it:

```
%MySecondMacro(data=sashelp.class,HowMany=3);
```

Here is the entire result:

```
Listing of First 3 observations in sashelp.class

Obs     Name      Sex    Age    Height    Weight

  1    Alfred      M      14     69.0      112.5
  2    Alice       F      13     56.5       84.0
  3    Barbara     F      13     65.3       98.0
```

If you precede the macro invocation statement with

```
OPTIONS MPRINT;
```

you will find this in your SAS log:

```
45   options mprint;
46   %MySecondMacro(data=sashelp.class,HowMany=3);
MPRINT(MYSECONDMACRO):   options nocenter nodate nonumber;
MPRINT(MYSECONDMACRO):   options obs=3;
MPRINT(MYSECONDMACRO):   title "Listing of First 3 observations in sashelp.class";
MPRINT(MYSECONDMACRO):   proc print data=sashelp.class;
MPRINT(MYSECONDMACRO):   run;

NOTE: There were 3 observations read from the data set SASHELP.CLASS.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

MPRINT(MYSECONDMACRO):   options obs=max;
```

MySecondMacro is a very simple example of using the macro facility to customize the processing, in this case, simply to limit the amount of output. You can make the customization capabilities as powerful as you wish and need, of course. In particular, macro processing can also enable you to customize the formatting of your output.

Before we get into more sophisticated macro capabilities, let's look at something basic and an essential part of the macro environment.

MACRO VARIABLES (a.k.a., SYMBOLIC VARIABLES):

The simplest macro language tools and how to use them

Macro language relies on the use of symbolic variables, also known as macro variables. They differ from SAS variables by being temporary entities whose values never reside permanently on disk.

Macro variable names can be up to 32 characters, a mixture of alphabetic letters, digits, and underscores. They must start with either a letter or an underscore. That is how they look in a statement which defines them or is used to create them.

Here is an example of creating and assigning a value to a macro variable:

```
%let MyFirstMacVar = 18;
```

The value stored for a macro variable stored by SAS in its temporary symbol table for your SAS session or batch program begins with the first non-blank character after the = sign and ends with the last non-blank character before the semi-colon. I.e., the same result as that from using the statement above would be stored by using this:

```
%let MyFirstMacVar =       18       ;
```

Here is a slightly more complicated example:

```
%let MySecondMacVar = LeRoy Bessler;
```

Imbedded blanks, like that between the parts of my name, are preserved when the value is stored by SAS in its temporary symbol table for your SAS session or batch program.

Here are ways to store leading or trailing blanks, if you need them:

```
%let MyThirdMacVarHas1LeadingBlank = %str( AfterOneBlank);
```

```
%let MyFourthMacVarHas1TrailingBlank = %str(BeforeOneBlank );
```

The %str function preserves any number of leading or trailing blanks.

To see whether there are leading or trailing blanks on a macro variable, run %put LLL&SomeMacVar.RRR where the dot is essential, and inspect the SAS log. There will be more discussion of that dot later.

After macro variables have been stored in the symbol table, with %let or any of the others ways to create them (other ways will explained shortly), you need a way to be able to retrieve them inside your program later.

Here is an example of retrieving a macro variable, similar to what you have seen in one of the macros above.

If I wanted my name to appear in a report footnote, I would use

```
footnote "Report Creator: &MySecondMacVar";
```

at the bottom of the report, I would see

```
Report Creator: LeRoy Bessler
```

Tip: If you need to retrieve a macro variable in a string that must be enclosed in quotes for SAS to use it, then the string must be enclosed in double quotes.

If I used use single quotes in the example above, as in

```
footnote 'Report Creator: &MySecondMacVar';
```

at the bottom of the report, I would see

```
Report Creator: &MySecondMacVar
```

Here is another example of retrieving and using a macro variable.

If I wanted to retrieve a list of names of students older than 13 in SASHELP.CLASS, I would use

```
%let MyFirstMacVar = 13;

data work.OlderThanThirteen(keep=Name);
set sashelp.class(keep=Age Name where=(Age GT &MyFirstMacVar));
run;
```

Tip: To use a macro variable in a string where it is concatenated with constant text (letters, numbers, a dot), the reference to the macro variable must be followed by a dot to terminate it. If concatenated with a dot, the reference must initially include two dots at the point, one as the terminator and the other as the survivor.

If you want to end up with, say, WebPageX.html, by assigning %LET DesiredWebPage = WebPageX prior to an ODS HTML statement, then that statement must be of this form:

```
ODS HTML PATH="SomePath" (url=none) BODY=&DesiredWebPage..html;
```

which will correctly resolve to:

```
ODS HTML PATH="SomePath" (url=none) BODY=WebPageX.html;
```

Without the extra dot, the resolution would be

```
ODS HTML PATH="SomePath" (url=none) BODY=WebPageXhtml;
```

SAS WOULD actually create output, but in the output folder (at SomePath) there would be a file with filename “WebPageXhtml” with filetype “File”, rather than filetype “HTML Document”. If anyone tried to open it, a prompt would come up, which says, ”Choose the program you want to use to open this file”. Not the desired outcome.

If you want to provide a prefix to a partially formed string, you need to provide only one dot. E.g., submitting

```
%let FirstPartOfMyFirstName = Le;
%let MyFullFirstName = &FirstPartOfMyFirstName.Roy;
%put MyFullFirstName is &MyFullFirstName;
```
will display this in the SAS log:

```
MyFullFirstName is LeRoy
```

If you want to produce a fully formed string out of two macro variables with no imbedded blank between them, you must provide no blank, and, of course, no dot, between them. E.g., submitting

```
%let FirstPartOfMyFirstName = Le;
%let SecondPartOfMyFirstName = Roy;
%let MyFullFirstName = &FirstPartOfMyFirstName&SecondPartOfMyFirstName;
%put MyFullFirstName is &MyFullFirstName;
```

will display this in the SAS log:

```
MyFullFirstName is LeRoy
```

Before looking at macros or programmatic ways to create macro variables, let's look at automatic macro variables. These are useful pieces of information that SAS makes available to your program as soon as it starts.

To get a list of them, submit

```
%put _automatic_;
```

In your SAS log you will get a long list of macro variables and their current assigned values. At start of a SAS session or SAS batch job, a lot of the values will be blank. During the session or job run, some of the values will change, depending on what has transpired. You can find the definition of the automatic macro variables in the SAS OnlineDoc.

Here are some examples, with non-blank values:

```
AUTOMATIC SYSDATE 12AUG12
AUTOMATIC SYSDATE9 12AUG2012
AUTOMATIC SYSDAY Sunday
AUTOMATIC SYSTIME 10:21
```

The limitation with the date and time information above is that it pertains to the date and time when the session or batch program started, not the current real date and time when you might want to record it in a report or an output observation. The input data being used to create your output might have changed between the start of your program and the time that it is actually retrieved by your program. How to create macro variables for the current date and current time will be included in a future update to this tutorial. (You can request a solution from the author via email.)

Other macro variables include:

SYSVER – the version of SAS running
SYSVLONG – includes the modification level of the version running
SYSSITE – the SAS site number for the copy of SAS that is open.
SYSUSERID – it can be important to be able to record the User ID that is running SAS and list that information in an output report or imbed it in an output data set that is being created.
SYSJOBID – this Windows or Unix Process ID is useful to know if your program is running on a server, you cannot stop it, and you need to call the server administrator to kill the process. You might have more than one process running, and you need to be sure that the right one is stopped, not the other(s).

MACRO LANGUAGE STATEMENTS THAT YOU CAN USE ANYWHERE

%let and %put can be used outside or inside of macros. Code that is outside of macros is called "open code". These are the only macro language statements that can be used in open code. %put allows you to display in the log any combination of real text and the text from resolution of macro variables, and was demonstrated above.

MACRO FUNCTIONS THAT YOU CAN USE ANYWHERE

%INDEX, %LENGTH, %SCAN, %SUBSTR, %UPCASE, %LEFT, and %TRIM are obvious analogues of INDEX, LENGTH, SCAN, SUBSTR, UPCASE, LEFT, and TRIM. (That is not a complete list.) These can be used in open code. That typically would happen as part of a %LET statement. An example appears below when in conjunction with using PROC SQL to prepare some macro variables, which are not immediately usable as delivered.

SCOPE OF MACRO VARIABLES: Global Versus Local

Macro variables that are created outside of a macro are automatically Global in scope. I.e., they can be retrieved by code anywhere, outside of any macros in the program or session, and inside of any macro.

Macro variables that are created by invocation of a macro (such as data= or HowMany= in the macro examples above) are automatically Local in scope.

Macro variables that are created by the processing of code generated by a macro are by default Local in scope. However, if the macro contains a statement such as

%global MyMacVarCreatedInThisMacro;

the macro variable will be Global in scope, and its value can be retrieved outside the macro by any subsequent processing. Successful references to &MacVarCreatedInSomeMacro can be made in open code that executes after the macro and can made by processing generated for other macros that are invoked after the creating macro.

If, for some reason, inside a macro you want to use and assign values to a macro variable name that is also used as a global macro variable outside of the macro, and you do not want to overlay an already existing global value, then you need to precede the first assignment reference inside the macro to that variable with a %local statement. If you never make such dual use inside and outside a macro, then you might never need to use the %local statement.

Anecdote: I decline to offer this as a tip, since I might so far have only been lucky. My intuitive practice, when encountering a “MACRO VARIABLE NOT RESOLVED” message in the SAS log, after verifying that my code to assign a value to the macro variable did run before the referencing code and that I do not have a typo problem, is to insert a %global statement before the assignment code. If the macro variable is assigned in open code, then it should be global by default. The problem is most likely to occur if you assign the macro variable inside a macro, but reference it outside the macro.

I do not pretend that my remarks are the be-all and end-all on this topic. For more about macro variables, and global versus local, of course consult the SAS Online Doc. For some expert opinion and dialogue about this topic, see, e.g., References 2 and 3.

OTHER WAYS TO ASSIGN VALUES TO MACRO VARIABLES: DATA Step and PROC SQL

So far, the %let statement, to assign macro variables either inside or outside macros, and the macro invocation parameters themselves have been demonstrated.

There are two other ways to create and assign macro variables either inside or outside macros.

Inside a DATA Step

```
data _null_;
length TallestStudent $ 8;
retain TallestStudent ' ' MaxHeight 0;
set sashelp.class end=LastOne;
if height GT MaxHeight then do;
  MaxHeight = height;
  TallestStudent = Name;
end;
if LastOne;
call symput('MaxHgt',MaxHeight);
call symput('Tallest',TallestStudent);
run;

options nocenter nodate nonumber;
title "&Tallest is Tallest Student with Height = &MaxHgt inches";
title2 "Got name and height macro variables using DATA Step and CALL SYMPUT";
proc print data=sashelp.class;
where height = &MaxHgt;
run;
```

Yes, I could instead get the maximum height with, e.g., PROC MEANS, but this example serves a demonstration purpose. Here is the result, with anomalies in the title:

```
Philip   is Tallest Student with Height =          72 inches
Got name and height macro variables using DATA Step and CALL SYMPUT

Obs     Name     Sex    Age    Height    Weight

 15    Philip     M      16      72        150
```

You can assert that the extra blank after “Philip” was forced by my choice of variable length, but, in a general case, you might not be able to easily determine the maximum length value by prior inspection of the input data. In any case, you need to define the length for TallestStudent to fit the longest student name—you will not know a priori which name will be that of the tallest student. If you don’t know the longest student name, you can safely make the length huge, since you will trim the result for the title.

The title is fixed with these code changes:

```
call symput('MaxHgt',trim(left(MaxHeight)));
call symput('Tallest',trim(left(TallestStudent)));
```

Here is the title result from the two code changes:

```
Philip is Tallest Student with Height = 72 inches
```

With PROC SQL

We can try to create the same report as above with PROC SQL with this code:

```
proc sql noprint;
select max(height) into: MaxHgt
from sashelp.class;
quit;

proc sql noprint;
select name into: Tallest
from sashelp.class
where height EQ &MaxHgt;
quit;

options nocenter nodate nonumber;
title "&Tallest is Tallest Student with Height = &MaxHgt inches";
title2 "Got name and height macro variables using PROC SQL NOPRINT";
proc print data=sashelp.class;
where height = &MaxHgt;
run;
```

As you can see below, the title suffers the same problem as first try above with CALL SYMPUT and DATA Step:

```
Philip   is Tallest Student with Height =       72 inches
Got name and height macro variables using PROC SQL NOPRINT

Obs     Name     Sex    Age    Height    Weight

 15    Philip     M      16      72        150
```

The solution is to insert two statements between the second PROC SQL NOPRINT step and the PROC PRINT step:

```
%let Tallest = %trim(%left(&Tallest));
%let MaxHgt  = %trim(%left(&MaxHgt));
```

Here is the title result from the two code insertions:

```
Philip is Tallest Student with Height = 72 inches
```

PROC SQL Solution Available with SAS V9.3 and Beyond

You can avoid the intermediate step of having to trim the macro variables with the PROC SQL “trimmed” option:

```
proc sql noprint;
select max(height) into :MaxHgt trimmed from sashelp.class;
quit;

proc sql noprint;
select name into :Tallest trimmed from sashelp.class where height EQ &MaxHgt;
quit;
```

The title result from the same PROC PRINT step as above is:

```
Philip is Tallest Student with Height = 72 inches
```

WHAT TO DO WHEN THE MACRO FUNCTION YOU NEED DOES NOT EXIST

In SAS Versions 9.2 and 9.3 there are a few dozen built-in macro functions, i.e., functions that can be used in a macro statement rather than in a DATA step. Most of them have no analogue in the Base SAS language, but %INDEX, %LENGTH, %SCAN, %SUBSTR, and %UPCASE are obvious analogues of INDEX, LENGTH, SCAN, SUBSTR, and UPCASE.

First, here is an example of using an available macro function. Suppose I want to break up a string into parts that are separated by one or more blanks. I want to break a string consisting of my full name into my first name and my last name. After running these statements,

```
%let MyName = LeRoy Bessler;

%let MyFirstName = %scan(&MyName,1);
%let MyLastName = %scan(&MyName,2);

%put MyName=&MyName MyFirstName=&MyFirstName MyLastName=&MyLastName;
```

the SAS log shows the following:

```
MyName=LeRoy Bessler MyFirstName=LeRoy MyLastName=Bessler
```

Regardless of the fact that it is not a typical need, now I want to remove the blank between my first name and last name. After running these statements,

```
%let MyName = LeRoy Bessler;

%let MyNameWithNoBlanks = %sysfunc(compress(&MyName));

%put MyName=&MyName MyNameWithNoBlanks=&MyNameWithNoBlanks;
```

the SAS log shows the following:

```
MyName=LeRoy Bessler MyNameWithNoBlanks=LeRoyBessler
```

MACRO FUNCTIONS FOR ARITHMETIC

I do not make any real applications of these arithmetic functions in this paper, but it is good for you to know about them for when you might need them. That you need a different function for non-integer values is not, in my opinion, at all intuitive to expect.

For integer arithmetic, the macro function is %EVAL. After running these statements,

```
%let ONE = 1;
%let TWO = 2;

%let THREE = %eval(&ONE + &TWO);

%put macro variable THREE = &THREE;
```

the SAS log shows the following:

```
macro variable THREE = 3
```

For non-integer arithmetic, the macro function is %SYSEVALF. After running these statements,

%let AreaOfCircleWithRadiusOfThree = %sysevalf(3.14159 * &THREE.**2); /* Note the . after &THREE */

%put AreaOfCircleWithRadiusOfThree = &AreaOfCircleWithRadiusOfThree;

the SAS log shows the following:

```
macro variable AreaOfCircleWithRadiusOfThree = 28.27431
```

CONDITIONAL PROCESSING WITH MACRO LANGUAGE

Another high-value capability is to use the macro facility for conditional processing. For DATA Step processing, the familiar Base SAS Language construct is:

IF <some condition>THEN DO;
< some statements >
END;
ELSE DO;
<some other statements >
END;

For use in a macro, there is the analogous construct where IF, THEN, DO, END, and ELSE are simply preceded with a % sign, and the construct can be used anywhere: in a DATA step, in a PROC step, or enclosing any collection of statements or steps.

Here is a first example of conditional processing.

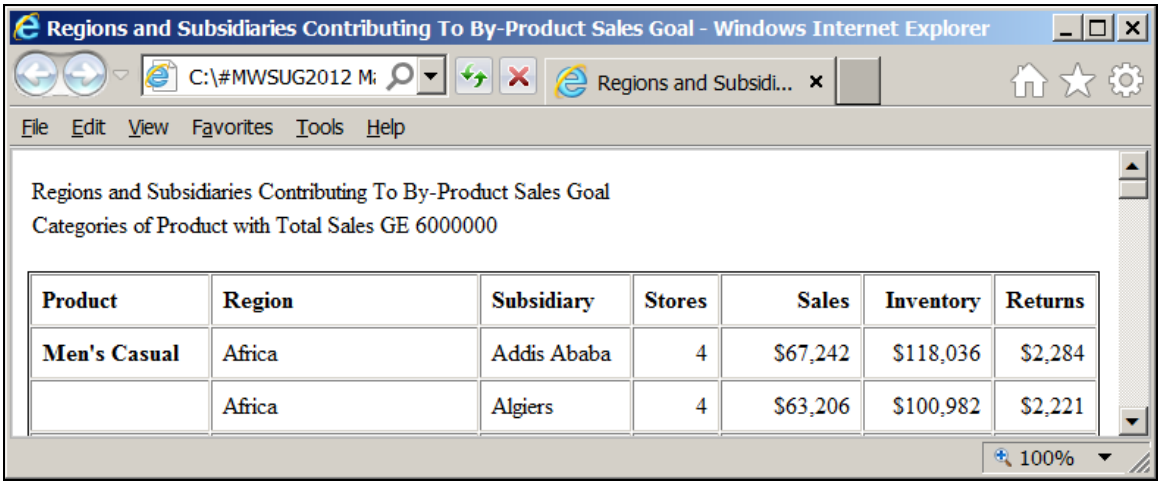
```
%macro DetailRptForTotalsOverMin(
data=
,ClassVarIsCharsNotNumeric=
,var=
,MinimumTotalOfInterest=0
,Title=
,RptPath=
,RptFile=);

proc summary data=&data nway;
class &ClassVarIsCharsNotNumeric;
var &var;
output out=TotalsOfVarByClass sum=TotalForClass;
run;

data _null_;
length ClassListForSelectUsingIN $ 32000;
retain ClassListForSelectUsingIN ' ';
set TotalsOfVarByClass end=LastOne;
if TotalForClass GE &MinimumTotalOfInterest
then do;
  ClassListForSelectUsingIN = trim(left(ClassListForSelectUsingIN)) ||
    ' "' || trim(left(&ClassVarIsCharsNotNumeric)) || '"';
  /* using double quotes on character values in case they contain single quotes */
  put ClassListForSelectUsingIN=; /* you can see this grow in length in the SAS log */
end;
if LastOne;
call symput('StringOfQuotedValuesForIN',trim(left(ClassListForSelectUsingIN)));
  /* above statement creates the key symbolic variable used to decide processing */
run;

options nocenter linesize=max;
ods noresults;
ods listing close;
ods html path="&RptPath" (url=none) body="&RptFile..html"(title="&Title")
    style=Styles.Minimal;
title "&Title";

%if %length(&StringOfQuotedValuesForIN) GE 3
```

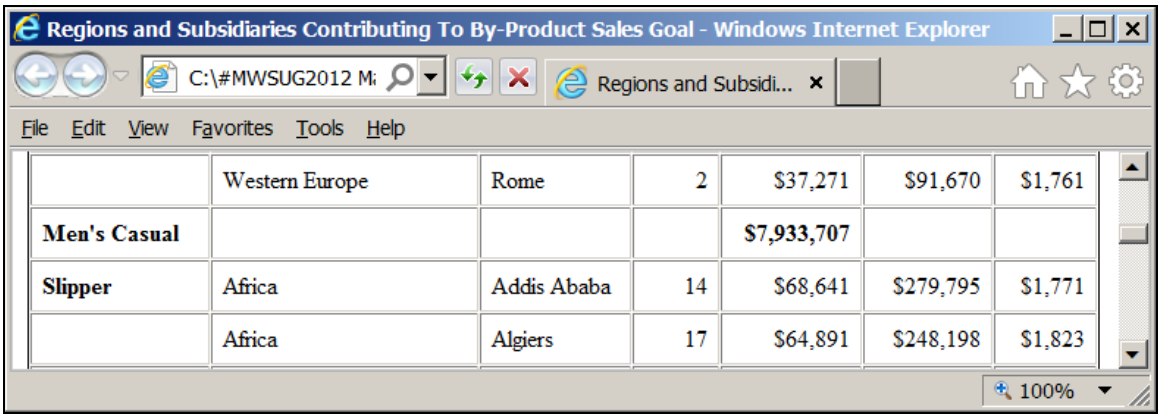
```
%then %do;
proc sort data=&data;
by &ClassVarIsCharsNotNumeric;
run;

title2
"Categories of &ClassVarIsCharsNotNumeric with Total &var GE &MinimumTotalOfInterest";
proc print noobs
     data=&data(where=(&ClassVarIsCharsNotNumeric IN (&StringOfQuotedValuesForIN)));
by &ClassVarIsCharsNotNumeric;
id &ClassVarIsCharsNotNumeric;
sum &var;
sumby &ClassVarIsCharsNotNumeric;
run;
%end;
%else %do;
data work.MessageToPrint;
Message =
"No Categories of &ClassVarIsCharsNotNumeric with Total &var GE
&MinimumTotalOfInterest";
run;

proc print data=work.MessageToPrint noobs label;
var Message / style = [background=yellow font_weight=Bold];
label Message='Nothing To Report';
run;
%end;

ods html close;
ods listing;

%mend  DetailRptForTotalsOverMin;

options MPRINT;
%DetailRptForTotalsOverMin(
data=sashelp.shoes
,ClassVarIsCharsNotNumeric=Product
,var=Sales
,MinimumTotalOfInterest=6000000
,Title=Regions and Subsidiaries Contributing To By-Product Sales Goal
,RptPath=C:\#MWSUG2012 Macros\Results
,RptFile=ShoeProductsWithAtLeastSixMillionDollarSales);
```

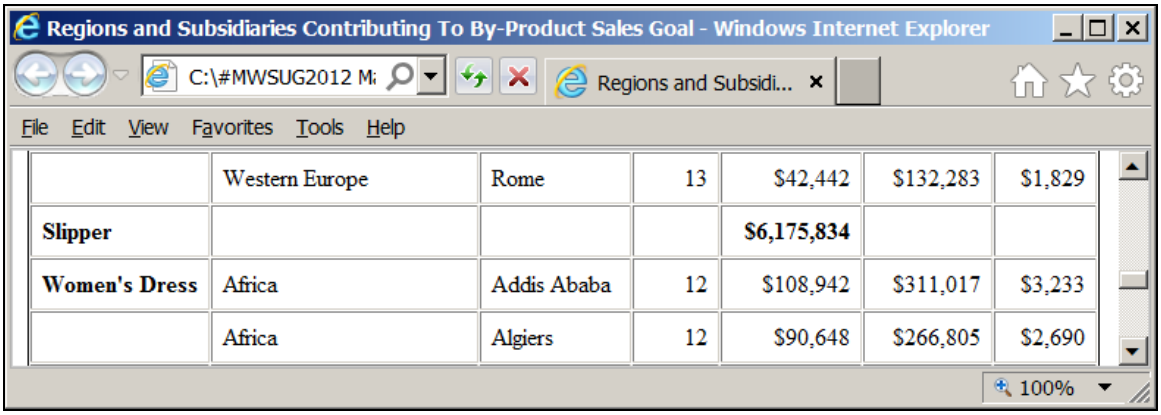
Here is what the final symbolic variable **`StringOfQuotedValuesForIN`** looks like when resolved, as shown in the SAS log:

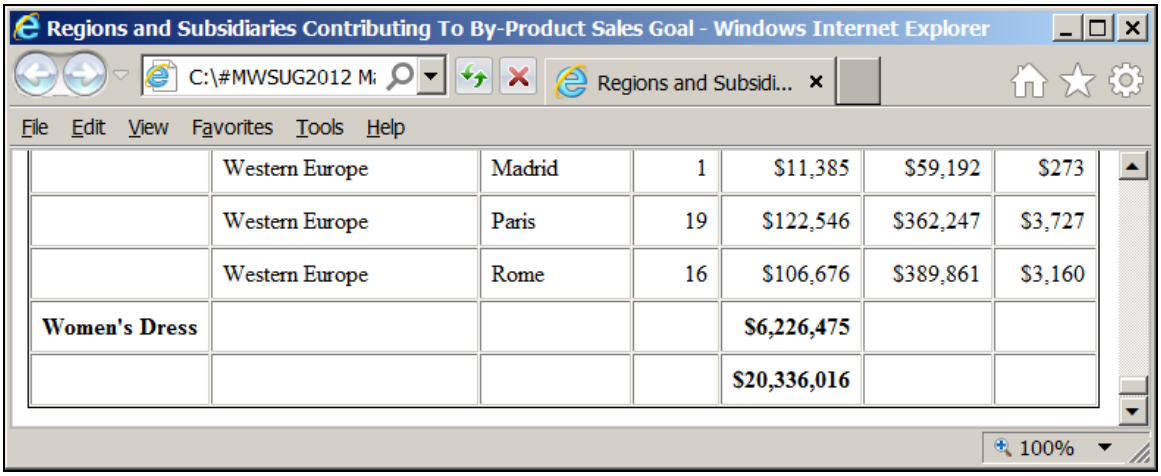
```
"Men's Casual" "Slipper" "Women's Dress"
```

Here are excerpts, scrolling from top to bottom, of the output created by the code invocation above:





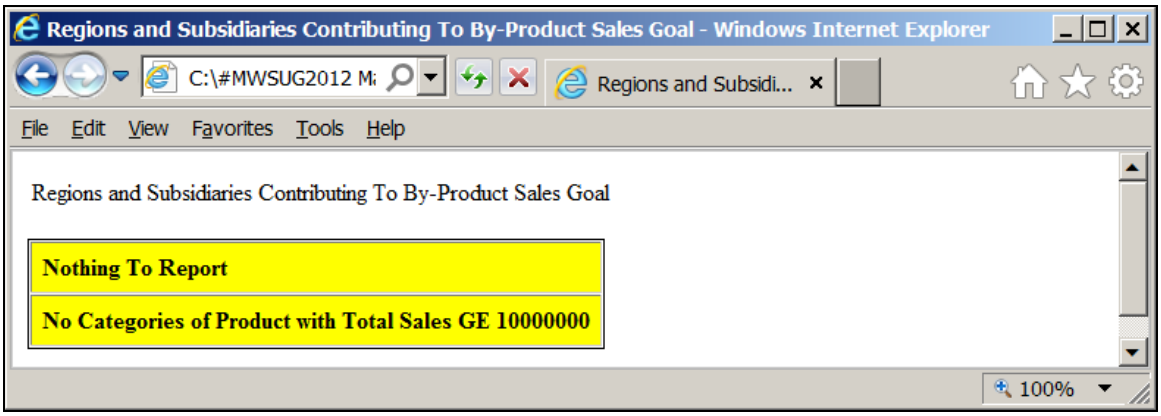




Now, by setting the Sales Dollars goal too high, to ten million dollars, as follows,

```
%DetailRptForTotalsOverMin(
data=sashelp.shoes
,ClassVarIsCharsNotNumeric=Product
,var=Sales
,MinimumTotalOfInterest=10000000
,Title=Regions and Subsidiaries Contributing To By-Product Sales Goal
,RptPath=C:\#MWSUG2012 Macros\Results
,RptFile=ShoeProductsWithAtLeastTenMillionDollarSales
);
```

the adverse report of failure looks like this:



FULL AUTOMATION TO ELIMINATE ANY NEED FOR PROGRAMMER INTERVENTION

In order to insulate the code from manual intervention, since something like a sales goal can be expected to change from time to time, the macro could be changed to read an external file controlled by the person empowered to set the goal or by some staff person at her/his direction. It is better to have a non-programmer be able to open a simple txt file and change a value than to require a program change. The only safe program change is no change. For such an implementation, the macro invocation code would look like the following:

```
%AutomatDetailRptForTotalsOverMin(
data=sashelp.shoes
,ClassVarIsCharsNotNumeric=Product
,var=Sales
,ControlFile=C:\#MWSUG2012 Macros\ControlFiles\MinimumSalesDollarsGoal.txt
,Title=Regions and Subsidiaries Contributing To By-Product Sales Goal
,RptPath=C:\#MWSUG2012 Macros\Results
,RptFile=ShoeProductsWithAtLeastSalesDollarsSpecifiedInControlFile
);
```

where PathToFile would typically be some location on the enterprise LAN, where the application has read access and the person managing the file would have ability to read and change the file.

The same output as shown in the first case above can be generated with this macro (where all changes are in blue):

```
%macro AutomatDetailRptForTotalsOverMin(
data=
,ClassVarIsCharsNotNumeric=
,var=
,ControlFile=
,Title=
,RptPath=
,RptFile=
);

data _null_;
infile "&ControlFile";
input NumericControlValue;
call symput('MinimumTotalOfInterest',trim(left(NumericControlValue)));
run;

< all code from the original DetailRptForTotalsOverMin macro goes here >

%mend  AutomatDetailRptForTotalsOverMin;

options MPRINT;

%AutomatDetailRptForTotalsOverMin(
data=sashelp.shoes
,ClassVarIsCharsNotNumeric=Product
,var=Sales
,ControlFile=C:\#MWSUG2012 Macros\ControlFiles\MinimumSalesDollarsGoal.txt
```

```
,Title=Regions and Subsidiaries Contributing To By-Product Sales Goal
,RptPath=C:\#MWSUG2012 Macros\Results
,RptFile=ShoeProductsWithAtLeastSalesDollarsSpecifiedInControlFile
);
```

COMMON PROCESSING FOR A LIST OF NUMBER-SUFFIXED MACRO VARIABLES

Often a SAS user wants to produce the same tabular report, graphic report, analysis, or data dump for a series of values of some classification variable. The same type of output for each customer, each product, each sales region, each time period, or whatever. SAS BY processing can accomplish this some of the time. But some processing needs are unsupported with BY processing, and for some output formatting objectives BY processing delivers the wrong format. In a case like these, a %DO loop can deliver ANYTHING that you can code for inside of it.

By now you are well familiar with macro variables of the form &SomeMacVarName, and understand the simple process of how they are resolved.

List processing of what I call “number-suffixed macro variables” is not obvious. For such list processing, inside the code block bounded by

```
%DO k = 1 %TO UpperLimit %by 1; /* UpperLimit is some integer,
                                   or could be a macro variable such as &Count */
```

at the top and bounded by

```
%END;
```

at the bottom, there will be references to macro variables of this form &&MacVarA&k, &&MacVarB&k, etc. As the index k works through the integers 1, 2, 3, etc. up to the upper limit, the reference &&MacVarA&k gets resolved in two steps.

For k = 1, the macro compiler first converts &&MacVarA&k to &MacVar1, and then it resolves &MacVar1 to whatever has been stored in the symbol table for MacVar1. The block of code in the %DO loop is then executed using the value for MacVar1. Analogous processing then occurs for k=2, k=3, etc.

Instead of looking at a real application, I will present a fake demonstration.

To serve as input, the code will create a SAS data set:

```
data work.ColorNamesAndRGBcolorCodes;
length Name $ 5 Code $ 8;
Name='Red';   Code='CXFF0000'; output;
Name='Green'; Code='CX00FF00'; output;
Name='Blue';  Code='CX0000FF'; output;
run;
```

There are two ways to prepare number-suffixed macro variables: DATA Step with CALL SYMPUT, or PROC SQL NOPRINT with SELECT . . . INTO.

After submitting this code:

```
data _null_;
set work.ColorNamesAndRGBcolorCodes end=LastOne;
call symput('ColorName'||trim(left(_N_)),trim(left(Name)));
call symput('ColorCode'||trim(left(_N_)),trim(left(Code)));
if LastOne;
call symput('NumberOfColors',_N_);
run;

%macro DisplayColorNamesAndColors;
%do i = 1 %to &NumberOfColors %by 1;
  %put The code for &&ColorName&i is &&ColorCode&i;
%end;
%mend  DisplayColorNamesAndColors;

%DisplayColorNamesAndColors;
```

the SAS log contains:

```
The code for Red is CXFF0000
The code for Green is CX00FF00
The code for Blue is CX0000FF
```

After submitting this code:

```
proc sql noprint;
select Name , Code, count(Name)
  into :ColorName1-:ColorName99999999, :ColorCode1-:ColorCode99999999, :NumberOfColors
  from work.ColorNamesAndRGBcolorCodes; quit;

%macro DisplayColorNamesAndColors;
%do i = 1 %to &NumberOfColors %by 1;
  %put The code for &&ColorName&i is &&ColorCode&i;
%end;
%mend  DisplayColorNamesAndColors;

%DisplayColorNamesAndColors;
```

the SAS log contains:

```
The code for Red is CXFF0000
The code for Green is CX00FF00
The code for Blue is CX0000FF
```

NOTE: In the general case, you do not know how many macro variables will be created by PROC SQL. It is harmless to use an absurdly large upper bound suffix (here, 99999999). Only the actual number of variables found will be used.

If the source values for the macro variables were not distinct in the input file, then preparing the macro variables is slightly more complicated. Consider the Region variable in SASHELP.SHOES. It occurs for multiple Products and multiple Subsidiaries. To get the list number-suffixed macro variables, DATA Step and CALL SYMPUT code is this:

```
proc sort data=sashelp.shoes(keep=Region) out=work.Regions nodupkey;
by Region;
run;

data _null_;
set work.Regions end=LastOne;
call symput('Region'||trim(left(_N_)),trim(left(Region)));
if LastOne;
call symput('NumberOfRegions',_N_);
run;

%macro DisplayRegionList;
%do i = 1 %to &NumberOfRegions %by 1;
  %put &&Region&i;
%end;
%mend  DisplayRegionList;

%DisplayRegionList;
```

The SAS log contains:

```
Africa
Asia
Canada
Central America/Caribbean
Eastern Europe
Middle East
Pacific
South America
United States
Western Europe
```

The PROC SQL code to produce the same result is:

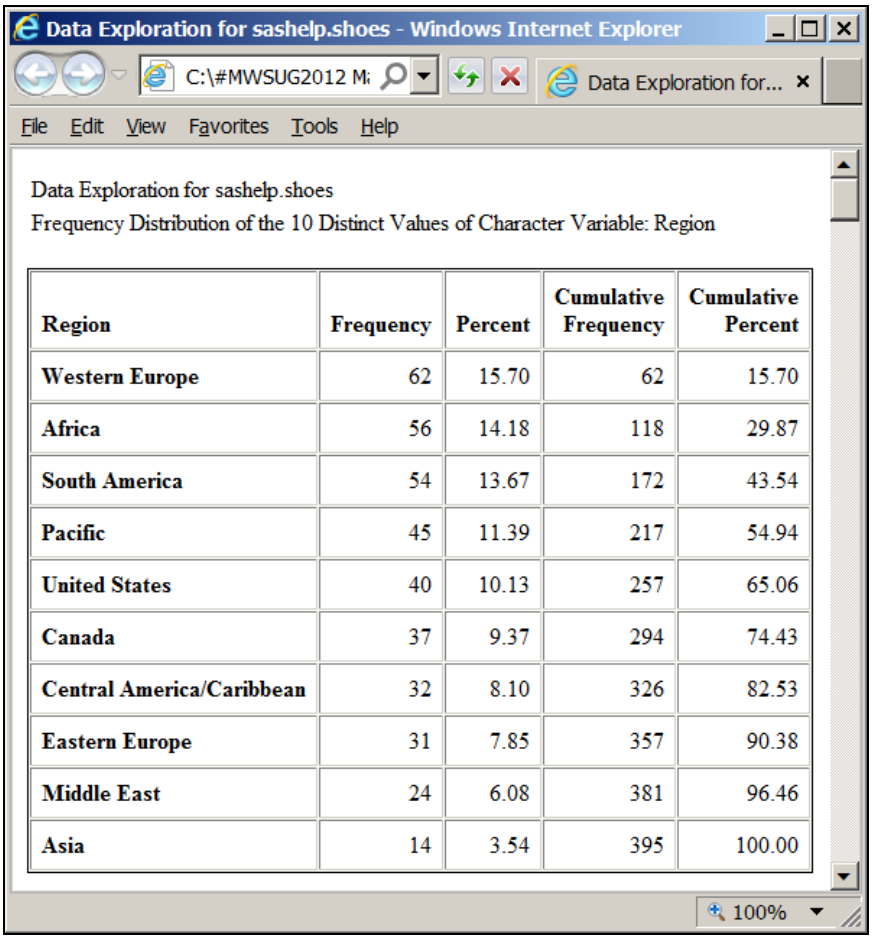
```
proc sql noprint;
select distinct Region , count(distinct Region)
  into :Region1-:Region999999999 , :NumberOfRegions
  from sashelp.shoes;
quit;

%macro DisplayRegionList;
%do i = 1 %to &NumberOfRegions %by 1;
  %put &&Region&i;
%end;
%mend  DisplayRegionList;

%DisplayRegionList;
```

NOTE: If the number of distinct values of the source for the macro variable is not fixed, and you do not know what its maximum value is, it is harmless to use an absurdly large upper bound suffix, such as 999999999. Only the actual number of distinct values found will be used.

Realistic Example of Widely Usable List Processing Macro You Can Use As Is or Enhance

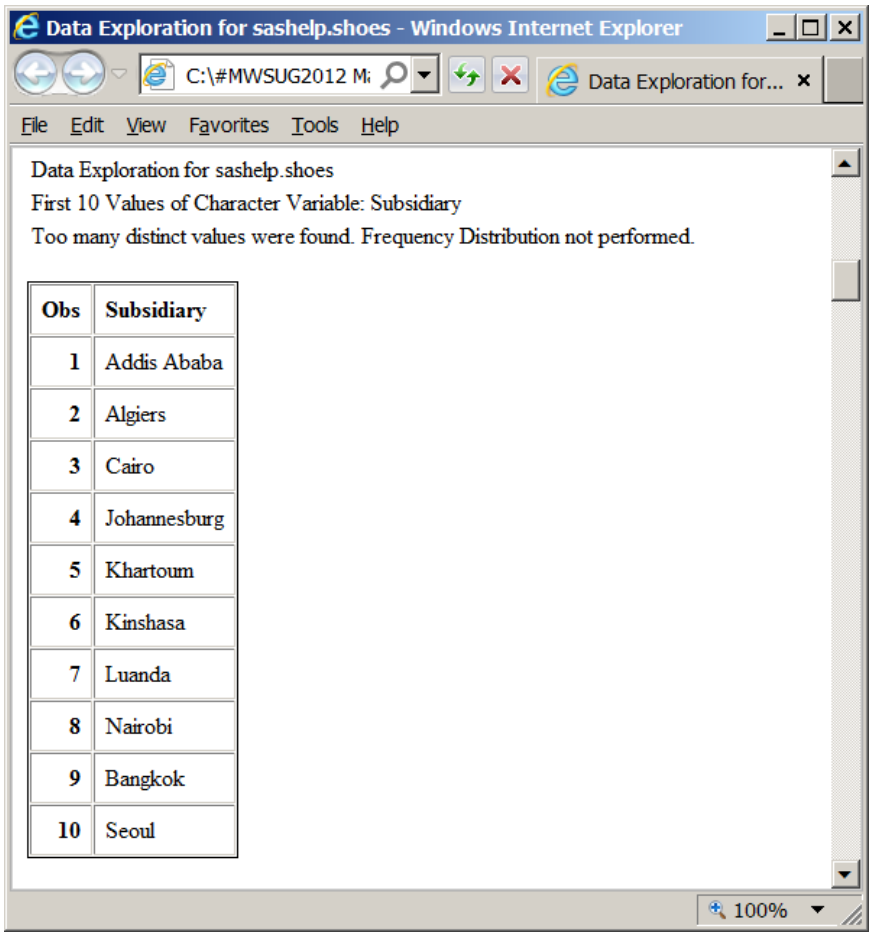
Suppose you want to quickly get an idea of what the actual values of numeric and character variables are like in an unfamiliar SAS data set. Below are a tool that you can use and example output for SASHELP.SHOES. It relies on use of the SAS utility data set DICTIONARY.COLUMNS which will give you information about the columns of all tables in all of the SAS data libraries that are currently allocated by your SAS program / SAS session. For your possible use in another situation, you should know that other column characteristics available, but not used here, include: length, format, label, and, if the data set is sorted on that column, which sort key it is (the data set could have multiple sort keys). First, let's look at the output, but with only a few excerpts, to avoid dealing with ALL of the variables in the SASHELP.SHOES data set.

For character variables, if the number of distinct values does not exceed what is assigned as macro parameter MaxDistinctForFreqAnalysis, then a frequency distribution is provided:



Region	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Western Europe	62	15.70	62	15.70
Africa	56	14.18	118	29.87
South America	54	13.67	172	43.54
Pacific	45	11.39	217	54.94
United States	40	10.13	257	65.06
Canada	37	9.37	294	74.43
Central America/Caribbean	32	8.10	326	82.53
Eastern Europe	31	7.85	357	90.38
Middle East	24	6.08	381	96.46
Asia	14	3.54	395	100.00

For the Subsidiary variable, there were more than 10 distinct values, and the output is this:



Data Exploration for sashelp.shoes - Windows Internet Explorer

C:\#MWSUG2012 M:

Data Exploration for...

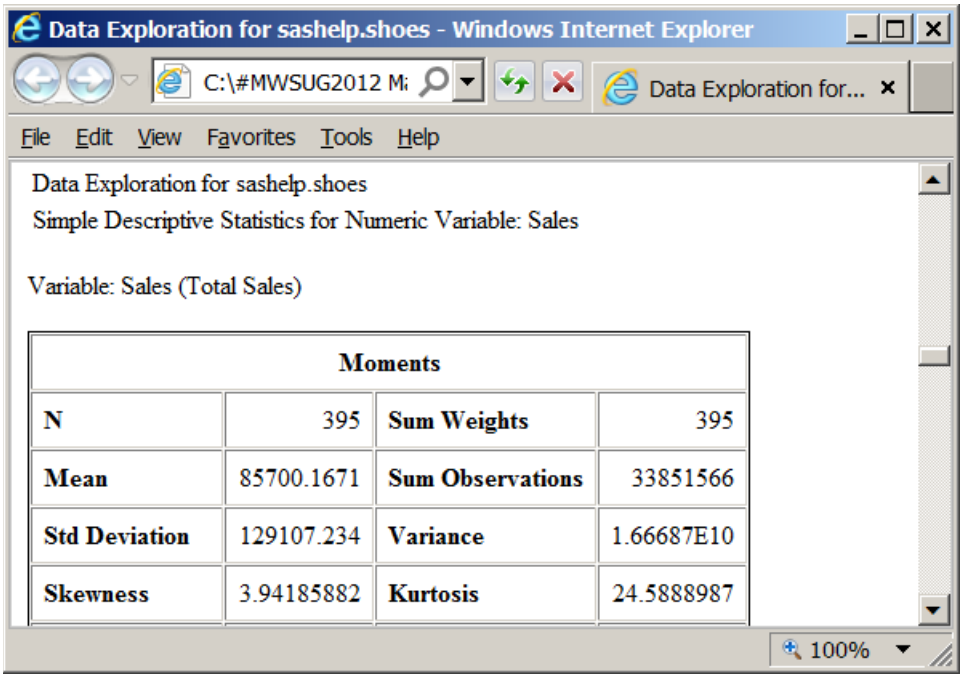
File Edit View Favorites Tools Help

Data Exploration for sashelp.shoes
First 10 Values of Character Variable: Subsidiary
Too many distinct values were found. Frequency Distribution not performed.

Obs	Subsidiary
1	Addis Ababa
2	Algiers
3	Cairo
4	Johannesburg
5	Khartoum
6	Kinshasa
7	Luanda
8	Nairobi
9	Bangkok
10	Seoul

100%

For any numeric variable, such as Sales, a PROC UNIVARIATE report is created. You could use PROC MEANS instead if your interest is less. Or add the request for a Histogram in your PROC UNIVARIATE code if you have Version 9.3 of Base SAS. Here is the top of the PROC UNIVARIATE output:



Data Exploration for sashelp.shoes - Windows Internet Explorer

C:\#MWSUG2012 M:

Data Exploration for...

File Edit View Favorites Tools Help

Data Exploration for sashelp.shoes
Simple Descriptive Statistics for Numeric Variable: Sales

Variable: Sales (Total Sales)

Moments			
N	395	Sum Weights	395
Mean	85700.1671	Sum Observations	33851566
Std Deviation	129107.234	Variance	1.66687E10
Skewness	3.94185882	Kurtosis	24.5888987

100%

Here are the Data Explorer macro code and an example of its invocation to produce the reports shown above:

```
%macro DataExplorer(
libname=
,datasetname=
,MaxDistinctForFreqAnalysis=
,RptPath=
,RptFileName=
);

proc sql noprint;
select type, name, count(name)
  into
  :type1-:type99999,
  :name1-:name99999,
  :CountOfColumns
  from dictionary.columns
  where
  libname EQ "%upcase(&libname)"
       and
  memname EQ "%upcase(&datasetname)";
quit;

OPTIONS NOCENTER;
ODS NORESULTS NOPROCTITLE;
ODS LISTING CLOSE;
ODS HTML PATH="&RptPath" (URL=NONE) GPATH="&RptPath" STYLE=Styles.Minimal GTITLE
    BODY="&RptFileName..html" (TITLE="Data Exploration for &libname..&datasetname");

title1 "Data Exploration for &libname..&datasetname";

%do i = 1 %to &CountOfColumns %by 1;

  %if &&type&i EQ num
  %then %do;

title2 "Simple Descriptive Statistics for Numeric Variable: &&name&i";
proc univariate data=&libname..&datasetname;
var &&name&i;
run;

  %end;

  %else %do;

proc sql noprint;
select count(distinct &&name&i) into :DistinctCharacterValues
  from &libname..&datasetname;
quit;

    %if &DistinctCharacterValues LE &MaxDistinctForFreqAnalysis
       %then %do;

title2 "Frequency Distribution of the &DistinctCharacterValues Distinct Values of
Character Variable: &&name&i";
proc freq data=&libname..&datasetname order=freq;
tables &&name&i / list missing missprint;
run;

    %end;
       %else %do;

options obs=&MaxDistinctForFreqAnalysis;
title2 "First &MaxDistinctForFreqAnalysis Values of Character Variable: &&name&i";
title3 "Too many distinct values were found. Frequency Distribution not performed.";
proc print data=&libname..&datasetname;
```

```
var &&name&i;
run;
options obs=max;

      %end;

  %end;

%end;

ODS HTML CLOSE;
ODS LISTING;

%mend  DataExplorer;

options MPRINT;

%DataExplorer(
libname=sashelp
,datasetname=shoes
,MaxDistinctForFreqAnalysis=10
,RptPath=C:\#MWSUG2012 Macros\Results
,RptFileName=DataExplorationForSASHELPdotSHOES
);
```

A SIMPLIFIED VERSION OF ONE OF MY FAVORITE APPLICATIONS OF SAS MACRO LANGUAGE or SOLUTIONS FOR A FINITE WORK DAY IN AN ERA OF INFORMATION OVERLOAD: SHOW THEM WHAT IS IMPORTANT . . . WITH: The Subsetted Ranking Report

For over a quarter century I have advocated and exploited use of the subsetted and ranked horizontal bar chart. You can find the code for a simplified version in Reference 4.

I always remember the wisdom of Jim White, an expert on printing, who said, “Let part stand for the whole.” I actually heard him several years after I became committed to the idea of trying to deliver only the most important. The most important can usually fit on one sheet of paper, and frequently, if not almost always, on one web page without having to scroll. This reminds me of the wisdom of Kenneth J. Wesley, my former staff who once counseled me, when I was working on a report for executive management, that “If it doesn’t fit on one page, they won’t read it.”

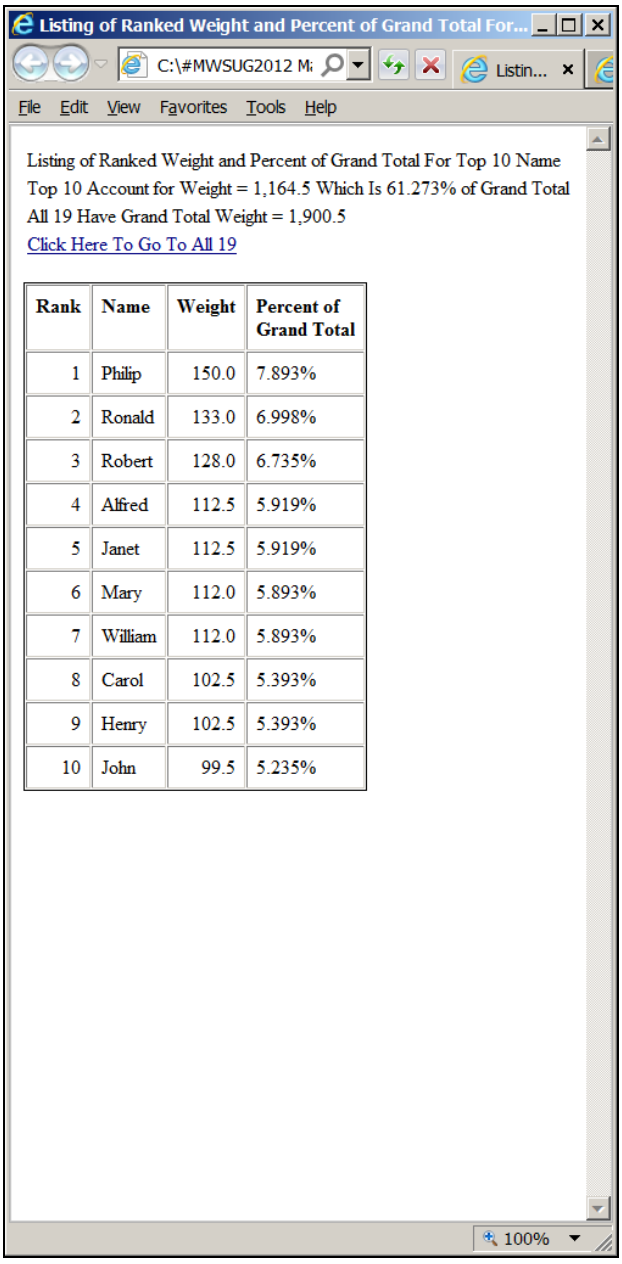
Many years ago I developed a macro that allowed the user to point to a data set and create a bar chart subsetted in any one of three ways: (1) Top N (where N was any integer); (2) all values above a cut-off; or (3) enough of the top values to account for the Top P Percent of the total measure of interest. Whereas I have long maintained that what will fit on a page, say, anywhere from the Top 10 to the Top 40 or 50, will usually account for 80 to maybe 99% of the total measure of interest, another good way to let part stand for the whole by subsetting is to stop reporting as soon as the chart bars account for as much of the total as you feel is important to show. YOU pick the percent target with Option 3 above.

However, during my latest consulting project, I decided that I wanted my client’s reporting system users to also be able to optionally look at the WHOLE list.

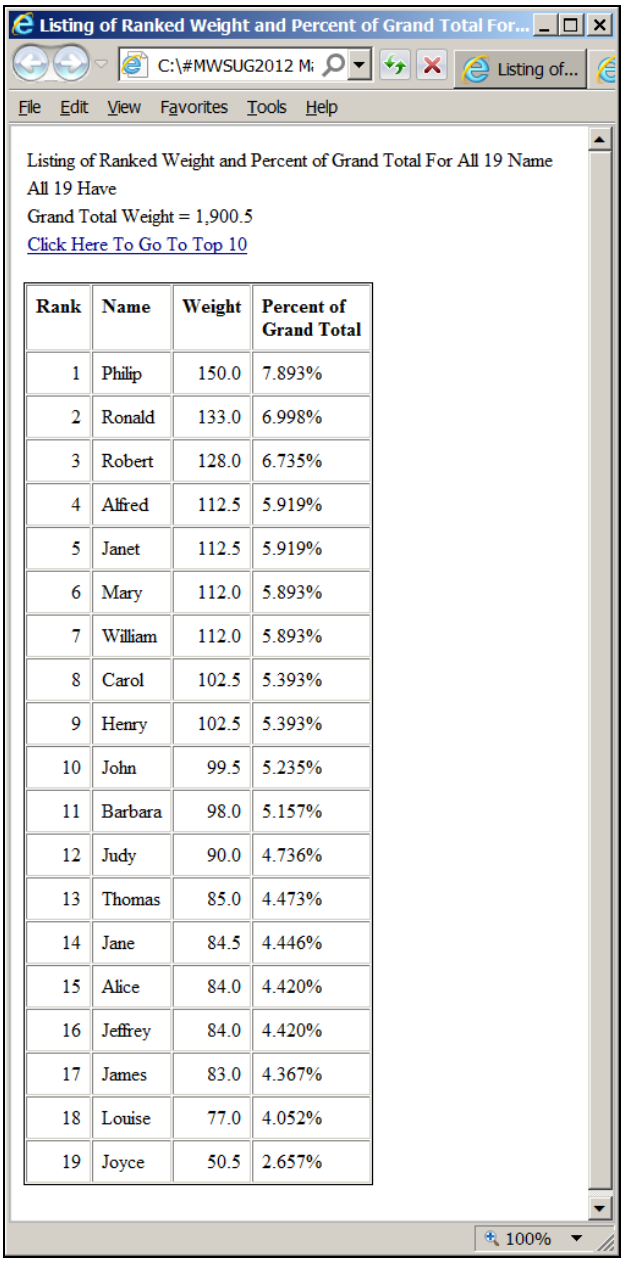
The macro that I am sharing in Reference 4 does NOT support that four-option capability. What was delivered for the client was actually a macro that not only created four versions of the ranking report, but also interlinked the four web graphs with hyperlinks—very cool and maximally convenient. In THIS macros tutorial, I do not get into the extra coding required for graphic output. The focus is on a tabular Subsetted Ranking Report.

In particular, the macro provided allows you to produce any subset of all of the data in one ranking report or a ranking report of ALL of the data, and to optionally web interlink the two reports. (In Reference 5, you can find the examples and code for an interlinked web package of ranked subsetted horizontal bar charts that include the full quartet of three ways of subsetting the data and the presentation of all of the data.)

On the next page is the pair of interlinked tabular reports:



Rank	Name	Weight	Percent of Grand Total
1	Philip	150.0	7.893%
2	Ronald	133.0	6.998%
3	Robert	128.0	6.735%
4	Alfred	112.5	5.919%
5	Janet	112.5	5.919%
6	Mary	112.0	5.893%
7	William	112.0	5.893%
8	Carol	102.5	5.393%
9	Henry	102.5	5.393%
10	John	99.5	5.235%



Rank	Name	Weight	Percent of Grand Total
1	Philip	150.0	7.893%
2	Ronald	133.0	6.998%
3	Robert	128.0	6.735%
4	Alfred	112.5	5.919%
5	Janet	112.5	5.919%
6	Mary	112.0	5.893%
7	William	112.0	5.893%
8	Carol	102.5	5.393%
9	Henry	102.5	5.393%
10	John	99.5	5.235%
11	Barbara	98.0	5.157%
12	Judy	90.0	4.736%
13	Thomas	85.0	4.473%
14	Jane	84.5	4.446%
15	Alice	84.0	4.420%
16	Jeffrey	84.0	4.420%
17	James	83.0	4.367%
18	Louise	77.0	4.052%
19	Joyce	50.5	2.657%

Here is the code used to produce the reports:

```
%macro SubsettedRankingReports(
data=
,RptPath=
,Title4_LinkToTableWithCount=
,ClassCountMax=
,ClassVar=
,ClassCountFormat=
,ClassMeasureVar=
,ClassMeasureFormat=);

PROC SUMMARY DATA=&data nway;
CLASS &ClassVar;
VAR &ClassMeasureVar;
OUTPUT OUT=Summed(drop=_type_ _freq_) SUM=;
RUN;
```

```
PROC SQL NOPRINT;
SELECT COUNT(&ClassVar) , SUM(&ClassMeasureVar)
  INTO
  :CountOfAllClasses    , :ClassMeasureGrandTotal
  FROM Summed;
QUIT;

PROC SORT DATA=Summed;
BY DESCENDING &ClassMeasureVar;
RUN;

%if %upcase(&ClassCountMax) NE ALL
%then %do;

OPTIONS OBS=&ClassCountMax;

%end;

DATA Selected;
LENGTH PercentOfGrandTotal $ 7;
SET Summed;
PercentOfGrandTotal = TRIM(LEFT(PUT(((&ClassMeasureVar
  / &ClassMeasureGrandTotal) * 100),6.3))) || '%';
Rank = _N_;
RUN;

OPTIONS OBS=MAX;

PROC SQL NOPRINT;
SELECT COUNT(&ClassVar)        , SUM(&ClassMeasureVar)
  INTO :CountOfSelectedClasses , :ClassMeasureSubTotal
  FROM Selected;
QUIT;

DATA _NULL_;
LENGTH ForSYMPUT ForSYMPUT1 ForSYMPUT2 8;
ForSYMPUT = &ClassMeasureGrandTotal;
CALL SYMPUT('GrandTotal',
  TRIM(LEFT(PUT(ForSYMPUT,&ClassMeasureFormat))));
ForSYMPUT = &ClassMeasureSubTotal;
CALL SYMPUT('SubTotal',
  TRIM(LEFT(PUT(ForSYMPUT,&ClassMeasureFormat))));
ForSYMPUT = &CountOfAllClasses;
CALL SYMPUT('CountOfAll',
  TRIM(LEFT(PUT(ForSYMPUT,&ClassCountFormat))));
ForSYMPUT = &CountOfSelectedClasses;
CALL SYMPUT('CountOfSelected',
  TRIM(LEFT(PUT(ForSYMPUT,&ClassCountFormat))));
ForSYMPUT1 = &ClassMeasureSubTotal;
ForSYMPUT2 = &ClassMeasureGrandTotal;
CALL SYMPUT('SubTotalPercentOfGrandTotal',
  TRIM(LEFT(PUT(( ForSYMPUT1 / ForSYMPUT2 ) * 100 , 6.3))));
RUN;

%let Title1_Prefix = Listing of Ranked &ClassMeasureVar and Percent of Grand Total
For;

%if %upcase(&ClassCountMax) EQ ALL
%then %do;
  %let Title1 = &Title1_Prefix.%str( All )&CountOfAll &ClassVar;
  %let Title2 = All &CountOfAll Have;
  %let Title3 = Grand Total &ClassMeasureVar = &GrandTotal;
  %let FileName = All&CountOfAll.&ClassVar;
%end;
%else %do;
  %let Title1 = &Title1_Prefix.%str( Top )&CountOfSelected &ClassVar;
```

```
  %let Title2 = Top &CountOfSelected Account for &ClassMeasureVar = &SubTotal Which Is
&SubTotalPercentOfGrandTotal.% of Grand Total;
  %let Title3 = All &CountOfAll Have Grand Total &ClassMeasureVar = &GrandTotal;
  %let FileName = Top&CountOfSelected.&ClassVar;
%end;

%if %length(&Title4_LinkToTableWithCount) NE 0 %then %do;
  %if %upcase(&Title4_LinkToTableWithCount) NE ALL
  %then %let Title4_LinkDescription = Top &Title4_LinkToTableWithCount;
  %else %let Title4_LinkDescription = All &CountOfAll;
  %let FileNameLinkTo = %sysfunc(COMPRESS(&Title4_LinkDescription.&ClassVar));
%end;

OPTIONS NOCENTER; FOOTNOTE;
ODS NORESULTS;
ODS LISTING CLOSE;
ODS HTML PATH="&RptPath" (URL=NONE) STYLE=Styles.Minimal
    BODY="&FileName..html" (TITLE="&Title1");

TITLE1 "&Title1";
TITLE2 "&Title2";
TITLE3 "&Title3";
%if %length(&Title4_LinkToTableWithCount) NE 0 %then %do;
TITLE4
  LINK="&FileNameLinkTo..html"
  "Click Here To Go To &Title4_LinkDescription";
%end;

PROC PRINT DATA=Selected NOOBS LABEL;
VAR Rank &ClassVar &ClassMeasureVar PercentOfGrandTotal;
LABEL PercentOfGrandTotal='Percent of Grand Total';
RUN;

ODS HTML CLOSE;
ODS LISTING;

%mend  SubsettedRankingReports;

options MPRINT;

%SubsettedRankingReports(
data=sashelp.class
,ClassCountMax=10
,RptPath=C:\#MWSUG2012 Macros\Results
,Title4_LinkToTableWithCount=ALL
,ClassVar=Name
,ClassCountFormat=comma5.
,ClassMeasureVar=Weight
,ClassMeasureFormat=comma11.1);

%SubsettedRankingReports(
data=sashelp.class
,ClassCountMax=All
,RptPath=C:\#MWSUG2012 Macros\Results
,Title4_LinkToTableWithCount=10
,ClassVar=Name
,ClassCountFormat=comma5.
,ClassMeasureVar=Weight
,ClassMeasureFormat=comma11.1);
```

CONCLUSION

SAS Macro Language can enable you to create reusable versions of your code, reducing the requirement to rewrite or manually revise code when needing to accomplish the same kind of task under slightly different conditions, and reducing probability of error by allowing you to change macro parameters rather than code itself. Furthermore, it can enable you to create adaptive processing that responds to changes in data, date, data date, data-date range, other

run-time matters, etc. in order to still meet your processing and output formatting objectives. Finally, to insulate your application from programmer intervention when control changes are required, the controls can be imbedded in external files, managed by administrative staff at management direction; and the external files can be read by the application code at start-up and used to initialize the variables that control subsequent processing.

REFERENCES AND OTHER RELATED READING

1. Bessler, LeRoy. "Intelligent Production Graphic Reporting Applications", *Proceedings of the Sixteenth Annual SAS Users Group International Conference*. Cary, NC, USA: SAS Institute Inc., 1991. The 1992 edition of this paper can be found online at http://www.sascommunity.org/sugi/SUGI92/Sugi-92-34%20Bessler.pdf
2. Carpenter, Art. "Carpenter's Complete Guide to the SAS Macro Language, Second Edition". Cary, NC, USA: SAS Institute Inc., 2004.
3. For "AN ONLINE LIVING BEST PRACTICES DISCUSSION AND DOCUMENT", started by Don Henderson and Art Carpenter, go to: http://www.sascommunity.org/wiki/Macro_Programming_Best_Practices:_Styles,_Guidelines_and_Conventions_Including_the_Rationale_Behind_Them
4. Bessler, LeRoy. "Get the Best Out Of SAS ODS Graphics and the SG (Statistical Graphics) Procedures: Communication-Effective Charts, Things That SAS/GRAPH Cannot Do As Well, and Macro Tools To Save Time and Avoid Errors", *MWSUG 2012 Conference Proceedings*. Minneapolis, MN: MidWest SAS Users Group, Inc., 2012. This paper can be found at http://www.mwsug.org/index.php/2012-proceedings.html
5. Bessler, LeRoy. "Data Visualization Power Tools: Expedite the Easy, Implement the Difficult, or Handle Big Data", *PharmaSUG 2013 Conference Proceedings*. Chicago, IL: Pharmaceutical SAS Users Group, 2013.
6. Burlew, Michele M. "*SAS Macro Programming Made Easy, Third Edition*". Cary, NC, USA: SAS Institute Inc., 2014.
7. Virgile, Robert. "*SAS Macro Language Magic: Discovering Advanced Techniques*". Cary, NC, USA: SAS Institute Inc., 2013.

AUTHOR INFORMATION

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

LeRoy Bessler PhD
Le_Roy_Bessler@wi.rr.com
Bessler Consulting and Research
Strong Smart Systems™
Visual Data Insights™
Mequon, Wisconsin, USA

A SAS user since 1978, Dr. LeRoy Bessler has shared his knowledge and experience with other users at conferences throughout the USA and in Montreal, London, Heidelberg, and Dublin. Though a SAS generalist with long experience in Base SAS, SAS macro language, and SAS tools for access to non-SAS data, his special interests include creation of unique tools to support the SAS BI server and its users, communication-effective visual communication and reporting, web information delivery, highly formatted Excel reporting, SAS/GRAPH, ODS, and Software-Intelligent Application Development for Reliability, Reusability, Extendibility, and Maintainability. If interested, send him an email request for an index to all of his SAS papers, presentations, and VIEWS Newsletter articles that are available via the internet.