Macros Made Simple

Tom Mannigel
Insyst Inc.
12800 Brair Forest #163
Houston, Texas 77077
(713) 975-8734
tmannigel@aol.com

This tutorial is designed to show you how to use Macro Processing to increase your programming proficiency the fast and easy way. To quickly teach SAS® programmers/users how to profit from Macro Processing, I have identified the most valueable features of Macro Processing by applying the Perot Principle (80/20 rule) to my work of over 7 years.

Alfred Perot an Italian economist back in the 1895 divide activities to two types: the vital few versus the trivial many. What he found was that in any activity: 20% of those activities account for 80% of the value. For example 20% of the customer of most company provide 80% of the business. I have identified the 20% of the Macro Processing that will give 80% of the value by categorizes Macro applications into easily understandable yet powerful types. Source Substitution using Macro Variables, Macro Statements and Macro Loops are the three types that will be discussed. Each of these three types are explained via meaningful and useful examples of how they can be used as potent programming tools.

Introduction:

Why Use Macros:

There are two very simple and important reasons to learn use Macros to write your SAS® programs. Using Macros will allow you to develop your programs faster and easier.

These are the reasons you'll program faster:

1. There's less to write.
2. Code can be reused.
3. Code is easier to change.

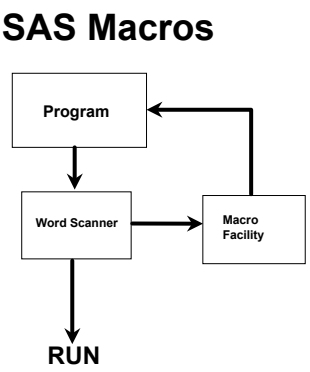
Who should use Macro Processing?

Everyone from the beginning programmer/user to the SAS® expert should use Macros.

Using Macro Processing for source substitution.

To appreciate the power of Macros we must first understand how Macros are processed by the SAS system. Take a look at Figure 1 and note how the source you generated is processed. First it is checked for Macro symbols such as & %. If the source has these symbols the SAS® system pass it to the Macro Facility where Macro variables and statements are substituted and then the new source is processed by SAS®.Its important to note that the program that actual runs has no macro symbols. It looks just like regular SAS code.

Figure 1. The Word Scanner and Macro Facility.



Source Substitution using Macro Variables:

Now let's begin with a simple but powerful example to see how Macro Variables work. In this example we are going to create a Macro Variable called &YR and use that variable to substitute for the year 94. Figure 2 shows the source we create with the Macro variables in it and shows the source after a pass through the Macro facility . Note the source that is pass to the SAS processor looks exactly as the source we would create but wherever &YR was found we now have 94. Note the Macro Variable is at the beginning of the program for easy changing.

Figure 2. An example of Source Substitution with Macro Variables.

```
%Let yr=94;
Proc Print
Data=library.data&yr;
  Title1 'American Report;
  Title2 " Year:19&yr";
```

Results after word scanner and Macro Facility.

```
Proc print
     Data=library.data94;
  Title1 "American Report";
  Title2 " Year:1994";
```

Note the need for double quotations in the second title if single quotations were used then the &YR would not be resolved.

Now lets take a look at a more powerful example of source substitution with Macro Variables. In this example(see Figure 3) we are going to use SAS® languages powerful "by" process capabilities, First we input the data with an infile and save the data in a SAS® library. Then use Proc Means to sum costs for 12 months for our company by division and department and final print the results. Figure 3 shows how we do that using two Macro Variables and also shows the results after the Macro Facility. Now the let's say we choose to do the same calculation and report except with a different company organization. In our new case we use region districts. To create this report all we do is simple change the %Let BY Statement to "%Let BY=company region district;" and our work is done.

Figure 3. More source substitute using Macro variables

```
 %Let Yr=94;
 %Let By=company division depart;
Libname library 'user Library';
```

```
Data Library.data&Yr;
Infile 'c:/company.dat';
Input year company division
     region depart
     district cost1-cost12;
     if year=&Yr;
Proc Sort;
      by &By;
Proc Means;
     by &By;
     var cost1-cost12;
     output sum= out=out;
Proc Print;
     by &By;
    sumby company;
    Title "Data for 19&Yr";
```

Results after Word Scanner and Macro Facility.

```
Data Library.Data94;
Infile 'c:\company.dat';
Input year company division
     region depart
     district cost1-cost12;
If Year=94;
Proc Sort;
     By company division
depart;
Proc Means;
     By company division
     depart;
     Var cost1-cost12;
     Output Sum= Out=Out;

 Proc Print;
      By company division
depart;
      Sumby company ;
      Title 'Data for 94';
```

Source Substitution Using Macro Statements.

Now lets say we want to create a report for 93 and 94 for this we move to our next application type and that is Macro Statements. Lets create a Macro Statements for our example in Figure 3. Figure 4 shows the results. Note the form :A %Macro with a "name" and the Macro is concluded with a %mend;. Then the Macro is called with a %"name". Now Figure 4 shows the above example except in macroized form.

Figure 4 Source Substitution using Macro Statements:

```
%Macro inputum;
Data library.data&Yr;
Infile 'c:/company.dat';
Input year company division
      region depart
      district
      Cost1-cost12 ;
 If year=&Yr;
 %mend;
%Macro sortsum;
 Proc Sort;
      by &By;
 Proc Means;
     by &By;
     Var cost1-cost12;
     output sum= out=out;
%mend ;

%Macro printum;
 Proc Print data=out;
      by &By;
      sumby company;
 Title "Data for 19&Yr";
%mend;

%Let Yr=94;
%Let By=company division
        depart;
%inputum
%sortum
%printum
%Let Yr=93;
%inputum
%sortum
%printum
```

After word scanner and Macro facility the results are exactly the same as Figure 3 except we now have two copies one for 93 and one for 94.

Note how compact and concise the code is and how easily changes can be made.

Passing Macro Variables to Macro Statements.

Figure 5 illustrates how to pass the Macro Variables to Macro Statement. Note there is no semicolon at the end of the Macro Call .

Figure 5 Three ways to pass Macro Variables to Macro Statements.

Positional:

```
%Macro inputum(yr);
Data library.data&yr;
Infile 'c:/company.dat';
Input year company division
    region depart
    district cost1-cost12;

 If year=&yr;
 %mend;
 %inputum(94)
```

Keyword:

```
%Macro sortsum(by=);
 Proc Sort; by &by;
 proc means;by &by;
     var cost1-cost12;
     output sum= out=out;
   %mend;

%sortum(by=company division
depart)
```

Default:

```
%Macro printum(yr=94,
       by=company division
          depart);
 Proc Print data=out;
      by &by;
       sumby company;
    Title "Data for 19&yr";

%printum() *Using Default;
%printum(yr=93) *Override;
```

Macro %Do Loops.

We are going to use Macro %Do loops to reduce the lines of code. Figure 6 shows marcoized form using Macro %Do loops. Note how compact this form is.

Figure 6. Using Macro %DO loops.

```
 %Macro Putit;
   %Do I=1 %To 6;
       profit&I=rev&I-
          staff&I-cost&I;
   %End;
%Mend;
%Putit
```

after the word scanner the above resolves to

```
profit1=rev1-staff1-cost1;
profit2=rev2-staff2-cost2;
profit3=rev3-staff3-cost3;
profit4=rev4-staff4-cost4;
profit5=rev5-staff5-cost5;
profit6=rev6-staff6-cost6;
```

For the above example we could have been used an array statement and a regular SAS do loop but remember regular SAS do loops can be used only in a data step.

The follow shows how to use a %Do for multiple data steps.

Figure 7 Using %Do loops outside a data step.

```
 %Macro Print23;
  %Do I=93 %to 94;
 Proc Print data=data&yr;
    Title "Data for 19&yr";
  %end;
%mend;
%print23
```

Results after the Word Scanner and Macro Facility:

```
Proc Print data=data93;
     Title "Data for 1993";
Proc Print data=data94;
     Title "Data for 1994";
```

%Do loops can also be used to loop Macro Statements. Figure 8 is how we put it all together to show the real power of SAS Macro Processing.

Figure 8 Putting it all together.

```
%Printyr(Begin,End);
%Do I= &Begin %To &End;
  %Inputum(&I)
  %Sortsum(Yr=&I)
  %Printum(Yr=&I,By=company
          depart division)
%End;
%Mend ;
%Printyr(70,94)
```

In this example we've created reports for 14 years of data and over 400 lines of code. Imagine how difficult it would be to change the unmacroized code and how easy the above example is to change.

Rules for Debugging SAS MACROS :

1. Debug a Macro by looking at the Macro output on the log .
1a. For Macro variables use Options symbolgen.
1b. For Macro Statements use Options mprint.
2. Make certain you called the Macro?
3. Are Macro variables correctly referenced?
4. %DO loops can not be in open code.
5. If the above does not solve the problem check your SAS® Macro manual.