

Abstract

At *LabOne, Inc.* we have used macros to effectively report on three business platforms using one generic report format. Utilizing macros, we are able to conditionally process pieces, sections and complete blocks of code. With one %LET statement we are able to change the complete look and feel of the end results. We use macro variables to pull certain data, keep specific variables, and set specific data sets. In addition, we use macros to assign macro variables to pull from specific libraries or datasets, and assign "where" criteria. This can be done for multiple business lines by changing one macro variable to coincide with that business.

This presentation will demonstrate utilization of macros to handle statistical reporting on multiple business services. We will show how to assign macros variables, effectively explain why and when to use a global statement and review correct troubleshooting procedures. We will begin with the basics of %LET and %PUT, touch briefly on symput and symget, and then move on to nested macros. We will also discuss using macros to pass titles into the report generation phase as we hand the report off to an inter/intranet environment. This presentation will include many real world production examples for those of you who are interested in bringing macros into your environment. While it may take a little time to learn and understand the SAS macro facility, the amount of time you save will more than make up for the time taken to learn.

Introduction

At LabOne, Inc. the Business Information Engineering Department has been using macros to streamline and simplify statistical reporting across three business units. By creating one "Generic" SAS job utilizing macros, we can access all three business services for different dates and different business locations with just a hand full of macro changes. While the "Generic" SAS code is macro intensive, it saves development time as we can create code for one client and re-use the same code for a different client, and possibly even a different client in a different business unit. This presentation will illustrate user-defined macro variables as well as system macro variables. We will even show you how to assign macros to conditionally run blocks of code. Then we will see how to pass macros into an intra/internet environment for reporting purposes.

User-Defined Macro Variables

User-defined macro variables are just that: they are macro variables defined by the user. I do need to point out that a macro is created as a character variable. If you plan on using macros in numeric values you will have to use some of the SAS functions to do so.

A user-defined macro variable may be as simple as:

```
%let Bdate=20010801;
```

You now have a user-defined macro variable. In a data set that has a date value, you can use this macro variable instead of changing code multiple times.

Example

```
%let Bdate=20010801;
```

```
Data dateinfo;  
  Set sathst.sa200108;  
  Where procdte eq "&Bdate";  
Run;
```

```
options nodate nonumber;  
Title "Output of History for &Bdate";  
Proc print data=dateinfo(obs=5);  
  var contnum procdte;  
Run;
```

Log Information

```
%let Bdate=20010801;  
92  
93 Data dateinfo;  
94   Set sathst.sa200108;  
95   Where procdte eq "&Bdate";  
96 Run;
```

NOTE: There were 10593 observations read from the data set SATHST.SA200108.

WHERE procdte='20010801';

NOTE: The data set WORK.DATEINFO has 10593 observations and 823 variables.

NOTE: Compressing data set WORK.DATEINFO decreased size by 92.54 percent.

Compressed is 66 pages; un-compressed would require 885 pages.

NOTE: DATA statement used:

real time	6.30 seconds
cpu time	4.51 seconds

```
96 !
```

```
97
```

```
98 options nodate nonumber;  
99 Title "Output of History for &Bdate";  
100 Proc print data=dateinfo(obs=5);  
101 var specid procdte;  
102 Run;
```

NOTE: There were 5 observations read from the

Macro Mania
Chris O'Brien, LabOne, Inc., Lenexa, KS

data set WORK.DATEINFO.

NOTE: PROCEDURE PRINT used:

real time 0.02 seconds
cpu time 0.01 seconds

options nodate nonumber;

Title "Output of History for &Bdate";

Proc print data=dateinfo(obs=5);

var specid procdte;

footnote "Report Created on &SYSDAY,
&SYSDATE9";

run;

Output

Output of History for 20010801

<i>Obs</i>	<i>SPECID</i>	<i>PROCDATE</i>
<i>1</i>	<i>1431540</i>	<i>20010801</i>
<i>2</i>	<i>1431541</i>	<i>20010801</i>
<i>3</i>	<i>1811500</i>	<i>20010801</i>
<i>4</i>	<i>1933174</i>	<i>20010801</i>
<i>5</i>	<i>2326298</i>	<i>20010801</i>

Output of History for 20010801

Obs	SPECID	PROCDATE
1	1431540	20010801
2	1431541	20010801
3	1811500	20010801
4	1933174	20010801
5	2326298	20010801

When using a macro variable in a condition statement comparing character data, you must enclose it in double quotation marks; instead of single quotation marks, otherwise it is handled as a string.

Data dateinfo;

2 Set sathst.sa200108;

3 Where procdte eq '&Bdate';

4 Run;

NOTE: There were 0 observations read from the data set SATHST.SA200108.

WHERE procdte='&Bdate';

NOTE: The data set WORK.DATEINFO has 0 observations and 823 variables.

NOTE: DATA statement used:

real time 3.52 seconds

cpu time 3.08 seconds

There you have it – a simple yet functional user-defined macro variable.

We can also use system-defined macro variables to include information in footnotes and headers. Here is a brief list of system-defined macro variables.

SYSDATE	14AUG01
SYSDATE9	14AUG2001
SYSDAY	Tuesday
SYSTIME	11:43

Example

Report Created on Monday, 13AUG2001

Macro Facility Functions

At LabOne, Inc., we utilize user-defined macro variables extensively. We store our data in the data warehouse in a monthly format (yyyymm). In the previous example, we hard-coded the set statement to sathst.sa200108.

So let's build a macro variable to set our monthly dataset, but let's use the Bdate macro variable that we have already assigned. With the SAS macro facility, you are able to use functions similar to base SAS such as substring(substr), right, left and trim. We will substring the BDATE macro to end up with a year/month macro and call it DSDATE.

%Let Bdate=20010801;

%LET DSDATE=%substr(&bdate,1,6);

This is a good time to tell you about some of the debugging/informational tools of the SAS Macro facility. Some of the most useful tools we've found are %PUT, symbolgen and mprint. These tools put information in the log for ease of coding. To utilize mprint and symbolgen, add them to the options statement as mprint|nomprint, and symbolgen|nosymbolgen. The mprint option will be used when creating and calling macros that we will discuss later.

%PUT "This is the requested date &BDATE so we need the &DSDATE dataset";

"This is the requested date 20010801 so we need the

Macro Mania
Chris O'Brien, LabOne, Inc., Lenexa, KS

200108 dataset"

Example of symbolgen

```
102 options symbolgen;
103 %Let Bdate=20010801;
104 %LET DSDATE=%substr(&bdate,1,6);
SYMBOLGEN: Macro variable BDATE resolves to 20010801
105 %PUT "This is the requested date &BDATE so we need the &DSDATE dataset"
105! ;
SYMBOLGEN: Macro variable BDATE resolves to 20010801
SYMBOLGEN: Macro variable DSDATE resolves to 200108
"This is the requested date 20010801 so we need the 200108 dataset"
```

Let's now look at using macros to conditionally execute blocks of code and to assign macro variables. LabOne has three business platforms: Insurance, Substance Abuse Testing and Clinical. With each business, we store our monthly data in separate libraries that have similar but different naming conventions. For example, Insurance will use '/ice/insurance/' with cyyyymm naming convention; Substance Abuse is '/ice/sat/' with sayyyymm and so on. We are going to look at how to create macros and conditionally pull data using one key macro variable.

At this point I need to explain the difference between a local and global macro variable. A local macro variable is only usable in the macro being processed at that time. If you create a macro variable within a macro, once that macro has completed execution any macro variables created within that macro are not available to be used. However, by using a global statement you make a local macro variable available to the entire job.

Code Example:

options symbolgen mprint;

```
%Let Bdate=20010801;
%let dset=SA;
%LET DSDATE=%substr(&bdate,1,6);

%PUT "This is the requested date &BDATE so we need the &DSDATE dataset";

%macro pulldata;
  %if &dset=SA %then %do;
    %let hist=SATHST;
```

```
    %let bsi=Substance Abuse Testing;
  %end;
  %if &dset=C %then %do;
    %let hist=INSHST;
    %let bsi=Insurance;
  %end;
  %if &dset=CL %then %do;
    %let hist=CLSHST;
    %let bsi=Clinical;
  %end;
%mend pulldata;
%pulldata
run;
```

```
Data &dset&dsetdate;
Set &hist..&dset&dsetdate;
Where procdat eq "&Bdate";
Run;
```

Log OutPut:

```
156 options symbolgen mprint;
157 %Let Bdate=20010801;
158 %let dset=SA;
159 %LET DSDATE=%substr(&bdate,1,6);
SYMBOLGEN: Macro variable BDATE resolves to 20010801
160 %PUT "This is the requested date &BDATE so we need the &DSDATE dataset"
160! ;
SYMBOLGEN: Macro variable BDATE resolves to 20010801
SYMBOLGEN: Macro variable DSDATE resolves to 200108
"This is the requested date 20010801 so we need the 200108 dataset"
161 %macro pulldata;
162   %if &dset=SA %then %do;
163     %let hist=SATHST;
164     %let bsi=Substance Abuse Testing;
165   %end;
166   %if &dset=C %then %do;
167     %let hist=INSHST;
168     %let bsi=Insurance;
169   %end;
170   %if &dset=CL %then %do;
171     %let hist=CLSHST;
172     %let bsi=Clinical;
173   %end;
174 %mend pulldata;
175 %pulldata
```

Macro Mania
Chris O'Brien, LabOne, Inc., Lenexa, KS

SYMBOLGEN: Macro variable DSET resolves to SA

SYMBOLGEN: Macro variable DSET resolves to SA

SYMBOLGEN: Macro variable DSET resolves to SA

176 run;

SYMBOLGEN: Macro variable DSET resolves to SA

SYMBOLGEN: Macro variable DSDATE resolves to 200108

177 Data &dset&dsdate;

178 Set

WARNING: Apparent symbolic reference HIST not resolved.

SYMBOLGEN: Macro variable DSET resolves to SA

SYMBOLGEN: Macro variable DSDATE resolves to 200108

178! &hist..&dset&dsdate;

-

22

201

ERROR: File WORK.SA200108.DATA does not exist.

ERROR 22-322: Syntax error, expecting one of the following: a name,

a quoted string, ,, END, KEY, KEYS, NOBS, OPEN, POINT, _DATA_, _LAST_, _NULL_.

ERROR 201-322: The option is not recognized and will be ignored.

179 Where procdat eq "&Bdate";

SYMBOLGEN: Macro variable BDATE resolves to 20010801

180 Run;

WARNING: Compression was disabled for data set WORK.SA200108 because

compression overhead would increase the size of the data set.

NOTE: The SAS System stopped processing this step because of errors.

WARNING: The data set WORK.SA200108 may be incomplete. When this step was

stopped there were 0 observations and 0 variables.

NOTE: DATA statement used:

real time 0.09 seconds

cpu time 0.01 seconds

What happened?? **WARNING: Apparent symbolic reference HIST not resolved.** This means that the macro variable HIST did not resolve. But we created it within the macro "pulldata"! Oh, remember what I said about local macro variables? Let's fix that by using a "%Global" statement.

%Global hist bsi;

%macro pulldata;

%if &dset=SA %then %do;

%let hist=SATHST;

%let bsi=Substance Abuse Testing;

%end;

%if &dset=C %then %do;

%let hist=INSHST;

%let bsi=Insurance;

%end;

%if &dset=CL %then %do;

%let hist=CLSHST;

%let bsi= Clinical;

%end;

%mend pulldata;

%pulldata

run;

Corrected Log:

%Global hist bsi;

183 %macro pulldata;

184 %if &dset=SA %then %do;

185 %let hist=SATHST;

186 %let bsi=Substance Abuse Testing;

187 %end;

188 %if &dset=C %then %do;

189 %let hist=INSHST;

190 %let bsi=Insurance;

191 %end;

192 %if &dset=CL %then %do;

193 %let hist=CLSHST;

194 %let bsi= Clinical;

195 %end;

196 %mend pulldata;

197 %pulldata

SYMBOLGEN: Macro variable DSET resolves to SA

SYMBOLGEN: Macro variable DSET resolves to SA

SYMBOLGEN: Macro variable DSET resolves to SA

198 run;

SYMBOLGEN: Macro variable DSET resolves to SA

SYMBOLGEN: Macro variable DSDATE resolves to 200108

199 Data &dset&dsdate;

Macro Mania
Chris O'Brien, LabOne, Inc., Lenexa, KS

SYMBOLGEN: Macro variable HIST resolves to SATHST
SYMBOLGEN: Macro variable DSET resolves to SA
SYMBOLGEN: Macro variable DSDATE resolves to 200108
200 Set &hist.&dset&dsdate;
201 Where procdate eq "&Bdate";
SYMBOLGEN: Macro variable BDATE resolves to 20010801
202 Run;

NOTE: There were 10593 observations read from the data set SATHST.SA200108.

WHERE procdate='20010801';

NOTE: The data set WORK.SA200108 has 10593 observations and 823 variables.

NOTE: Compressing data set WORK.SA200108 decreased size by 92.54 percent.

Compressed is 66 pages; un-compressed would require 885 pages.

NOTE: DATA statement used:

real time 8.42 seconds

cpu time 4.62 seconds

You may have noticed the two dots between the macro variable &hist and macro variable &dset in the set statement. This notation informs the SAS Macro facility that we need to place a single dot between the two macro variables.

**Set &hist.&dset&dsdate;
SATHST.SA200108**

Great, now by changing one macro variable we are able to pull data from three possible libraries.

Error Handling and Nested Macros

By using macros we are also able to stop a production job if encounters an empty dataset. This prevents the on-call person from receiving an error call that is due to missing data. The SAS Macro facility allows the use of "if/then/else" statements. By containing the code inside a macro we are able to use "if/then" logic to stop a job when we suspect a dataset may or may not be populated. The "if/then" logic needs to be preceded by the "%" symbol.

One of the system macro variables that has not yet been mentioned is the &SQLOBS. This variable is populated by the last SQL procedure that has run. &SQLOBS contains the number of observations that reside in the table the procedure created.

Example

**%macro runprod;
options symbolgen mprint macrogen;**

**%Let Bdate=20010801;
%let dset=SA;
%LET DSDATE=%substr(&bdate,1,6);**

%PUT "This is the requested date &BDATE so we need the &DSDATE dataset";

%Global hist bsi;

**%macro pulldata;
%if &dset=SA %then %do;
%let hist=SATHST;
%let bsi=Substance Abuse Testing;
%end;
%if &dset=C %then %do;
%let hist=INSHST;
%let bsi=Insurance;
%end;
%if &dset=CL %then %do;
%let hist=CLSHST;
%let bsi=Clinical;
%end;
%mend pulldata;
%pulldata
run;**

**Data &dset&dsdate;
Set &hist.&dset&dsdate;
Where procdate eq "&Bdate";
Run;**

**proc sql;
create table runprod as
select *
from &dset&dsdate;
quit;**

**%if &sqlobs
eq 0 %then %do;
%put "We have &sqlobs obs the job will stop";
endsas
%end;
%else %do;
%put "We have &sqlobs obs the job will
continue";
%end;**

**%mend runprod;
%runprod;**

The runprod macro allows us to use "if/then" logic in open code. We could have easily enclosed this logic inside of its

Macro Mania
Chris O'Brien, LabOne, Inc., Lenexa, KS

own macro such as the following example illustrates.

```
%macro stopper;
%if &sqlobs eq 0 %then %do;
    %put "We have &sqlobs obs the job will stop";
    endsas
%end;
%else %do;
    %put "We have &sqlobs obs the job will
continue";
%end;
%mend stopper;
%stopper;
```

It works both ways, giving the programmer options and allowing him/her to choose which will work best in a particular situation. Here is an example log.

Log Output:

```
189 proc sql;
190 create table runprod as
191 select *
192 from &dset&dsdate;
SYMBOLGEN: Macro variable DSET resolves to
SA
SYMBOLGEN: Macro variable DSDATE resolves
to 200108
NOTE: Compressing data set WORK.RUNPROD
decreased size by 92.54 percent.
Compressed is 66 pages; un-compressed would
require 885 pages.
NOTE: Table WORK.RUNPROD created, with
10593 rows and 823 columns.
```

```
193 quit;
NOTE: PROCEDURE SQL used:
    real time      2.09 seconds
    cpu time       2.03 seconds
```

```
194 %macro stopper;
195 %if &sqlobs
eq 0 %then %do;
196 %put "We have &sqlobs obs the job will stop";
197 endsas
198 %end;
199 %else %do;
200 %put "We have &sqlobs obs the job will
continue";
201 %end;
202 %mend stopper;
203 %stopper;
```

SYMBOLGEN: Macro variable SQLOBS resolves to 10593

SYMBOLGEN: Macro variable SQLOBS resolves to 10593

"We have 10593 obs the job will continue"

Symget/Symput

Symget and Symput are used to make data variables into a macro variable or vice versa. With a few lines of code we can take a macro variable and place it in a dataset.

Code Example:

```
%let appssn=55512!;
```

```
data datamac;
    appssn=symget(trim('appssn'));
run;
```

Print of datamac:

Macro variable turned to Data variable

Obs	appssn
1	'55512!'

This becomes very useful for handing variables from one environment to another. We have also found these tools to be useful when using HTML as an intra/internet coding tool. We have had instances where we wanted to do a wild card search on data from a web-based report. We found that HTML uses the “%” symbol as a special character. To get around this issue, we employed the “!” character in HTML and used SAS functions to translate it.

```
276 %let appssn=55512!;
277 %put This is how HTML likes wild cards
&appssn;
SYMBOLGEN: Macro variable APPSSN resolves to
55512!
```

This is how HTML likes wild cards 55512!

```
279 data _null_;
280 appssn=symget(trim('appssn'));
281 appssn=translate(appssn,'%','!');
282 appssn="'"||left(trim(appssn))||"'";
283 call symput('appssn',appssn);
284 run;
```

NOTE: DATA statement used:

<i>real time</i>	<i>0.00 seconds</i>
<i>cpu time</i>	<i>0.01 seconds</i>

***SYMBOLGEN: Macro variable APPSSN resolves to
'55512%'***

284!

***285 %put This is how SAS like wild cards &appssn;
This is how SAS like wild cards '55512%'***

With the use of forms in HTML, we are able to hand macros to the SAS server and utilize them for conditional sets, titles, and footnotes. This information can then be passed to the ODS system for output, or used in your company's chosen web-based output.

Conclusion

From the information that we have discussed, you should now be able to create user-defined macro variables. Conditionally pull data using your macro variable, print that information in the title, and use system-defined macro variables in the footnote. Using a macro, we showed how to use "if/then" logic to define macro variables and to conditionally step through code. We discussed the use of "%PUT", symbolgen and mprint, and why and when to use the global statement to make local macro variables available to the code. Finally, we demonstrated how symget and symput can be used to make macro variables available in a dataset and how to manipulate the macro variable.