

Beginning Macros

Prepared by



International SAS® Training and Consulting

Destiny Corporation
100 Great Meadow Rd Suite 601
Wethersfield, CT 06109-2379
Phone: (860) 721-1684 1-800-7TRAINING
Fax: (860) 721-9784
Email: info@destinycorp.com
Web: www.destinycorp.com
Copyright 2002

Macro variables

In this module we discuss the first of the two special characters - the ampersand (&).

When the SAS Supervisor sees an ampersand followed by a non-blank character, the macro facility is triggered. In turn, the macro facility determines the value for the macro variable and passes the value back on to the input stack.

If the macro facility fails to find the current value for a macro variable, the following message appears on the SAS Log:

Warning: Apparent symbolic reference is not resolved.

In version 5 systems this message is accompanied by the number 1301:

Warning 1301: Apparent symbolic reference is not resolved.

Strings enclosed within single blip quotes (') are always assumed to be one continuous string. Therefore, macro variable references within single quotes will NOT resolve. Tokens enclosed within double blip quotes (") are treated separately and thus macro variable references WILL resolve.

In release 5.18 and below, the facility needs to be enabled by:

OPTION DQUOTE;

In version 6 the system works as though DQUOTE is always enabled.

Predict the results of the following statements. Assume you are working on a version 6 system.

PROGRAM EDITOR

```
title 'Macro facility involves % and &';
title2 'Macro facility came as a result of much R&D';
title3 "Macro facility came as a result of much R&D";
title4 "Report prepared on &sysdate";
title5 'Report prepared on &sysday';
```

Automatic Macro Variables

When SAS is invoked, a set of Automatic macro variables is created; some of these are read-only, others are read-write. We have encountered two of these so far -&SYSDATE and &SYSDAY.

There are certain variations in the variables available in different versions of the systems as shown below. Note that these are the automatic macro variables available with the base product only.

Macro variables and their current values are stored in internal work areas called Symbol Tables. When the macro processor is trying to resolve a macro variable reference it will scan its symbol tables for a macro variable of that name and either retrieve its value and place that value upon the input stack, or the message

Warning: Apparent Symbolic reference¯o-variable not resolved.

will be issued.

When the SAS system is invoked (and assuming system options enable the macro facility) the GLOBAL SYMBOL TABLE is built. This holds most of the automatic macro variables.

Later we shall see other symbol tables created and deleted dynamically, how we can force variables into different tables and how the tables have a prescribed search order.

Creating macro variables

Within the macro facility and Data step language there are no fewer than 9 ways of creating a macro variable. We shall see all 9 methods throughout this course.

Let us begin with the simplest method, the %LET statement. The presence of a % followed by a non-blank character triggers the macro facility. A %LET tells the macro facility that a macro variable is to be defined. %LET can appear anywhere within a SAS program to define one macro variable at a time.

The form of the %LET statement is:

%LET macrovariable = value;

- value is optional:

%LET price=;

The macro variable price is created but takes a null value.

- trailing and leading blanks are ignored:

%LET cost= very cheap ;

creates a macro variable called cost with the value of very cheap. Note the embedded blank is included but no blanks before very or after cheap are included.

- excepting the case above, all characters are included as part of the macro variable including any quotes used.

```
%LET cost= 'very cheap' ;
```

creates a macro variable called cost with the value

```
'very cheap'
```

including all quotes and spaces. Note that you cannot do

```
%LET cost= 'very cheap' ;
```

This will give unbalanced quotation mark errors.

- inclusion of special characters in a macro variable value (except & or %) or leading or trailing blanks can be achieved by use of the %STR function:

```
%LET code=proc print;
```

```
%LET code2=%str(&code;run);
```

code1 takes the value of proc print

code2 takes the value of proc print;run;

- to include & or % as part of the string use %NRSTR:

```
%LET code=proc print;
```

```
%LET code2=%nrstr(&code;run);
```

code1 takes the value of proc print

code2 takes the value of &code;run;

- to inspect the variable's value simply use the %PUT statement to write the value of the Macro variable out to the log. Remember to reference it with an ampersand:

```
%PUT &code;
```

Writes the value of the macro variable code to the SAS Log.

PROGRAM EDITOR

```
%let station1=Paddington;
%put &station1;

%let station2=Clapham Junction;
%put &station2;

%let prefix1=St;
%let name1=Pancras;
%let station3=&prefix1 &name1;
%let suffix1=Street;
%let name2=Liverpool;
%let name3=Cannon;
%let station4=%str( Victoria );
%put &station4;

%let station5=&name2 &suffix1;
%put &station5;

%let station6=      &name3 &suffix1;
```

```
%put &station6;
```

```
%let station7='Tottenham Court Road';
%put &station7;
```

```
%let station8='Tottenham Court Road;Oxford Circus';
%put &station8;
```

```
%let station9=%nrstr(Highbury&Islington);
%put &station9;
```

```
%let stationa=%str(Chalfont&Latimer);
%put &stationa;
```

```
%let stationb=%str(Harrow&Wealdstone);
%put &stationb;
```

```
%let stationc="&station7";
%put &stationc;
```

These %LET and %PUT statements write the following Log:

LOG

NOTE: The PROCEDURE PRINTTO used 0.11 seconds.

MPRINT(PROGRAM): DM 'clear log; clear out';

SYMBOLGEN: Macro variable N resolves to m234

SYMBOLGEN: Macro variable STATION1 resolves to Paddington

Paddington

SYMBOLGEN: Macro variable STATION2 resolves to Clapham Junction

Clapham Junction

SYMBOLGEN: Macro variable PREFIX1 resolves to St

SYMBOLGEN: Macro variable NAME1 resolves to Pancras

SYMBOLGEN: Macro variable STATION4 resolves to Victoria

SYMBOLGEN: Some characters in the above value which were subject to macro quoting have been unquoted for printing.

Victoria

SYMBOLGEN: Macro variable NAME2 resolves to Liverpool

SYMBOLGEN: Macro variable SUFFIX1 resolves to Street

SYMBOLGEN: Macro variable STATION5 resolves to Liverpool Street

Liverpool Street

SYMBOLGEN: Macro variable NAME3 resolves to Cannon

SYMBOLGEN: Macro variable SUFFIX1 resolves to Street

SYMBOLGEN: Macro variable STATION6 resolves to Cannon Street

Cannon Street

SYMBOLGEN: Macro variable STATION7 resolves to Tottenham

Court Road'

'Tottenham Court Road'

SYMBOLGEN: Macro variable STATION8 resolves to Tottenham

Court Road;Oxford Circus '

'Tottenham Court Road;Oxford Circus'

```

SYMBOLGEN: Macro variable STATION9 resolves to
Highbury&Islington
SYMBOLGEN: Some characters in the above value which were
subject to macro quoting have been unquoted for
printing.
Highbury&Islington
WARNING: Apparent symbolic reference LATIMER not
resolved.
SYMBOLGEN: Macro variable STATIONA resolves to
Chalfont&Latimer
SYMBOLGEN: Some characters in the above value which were
subject to macro quoting have been unquoted for
printing.
WARNING: Apparent symbolic reference LATIMER not
resolved.
Chalfont&Latimer
ERROR: Symbolic variable name WEALDSTON must be 8 or
fewer
characters long.
SYMBOLGEN: Macro variable STATIONB resolves to
Harrow&Wealdstone
SYMBOLGEN: Some characters in the above value which were
subject to macro quoting have been unquoted for
printing.
ERROR: Symbolic variable name WEALDSTON must be 8 or
fewer
characters long.
Harrow&Wealdstone
SYMBOLGEN: Macro variable STATION7 resolves to
'Tottenham
Court Road'
SYMBOLGEN: Macro variable STATIONC resolves to
'"Tottenham
Court Road"'
'"Tottenham Court Road"'
MPRINT(PROGRAM): PROC PRINTTO;
MPRINT(PROGRAM): RUN;

```

Notes on the log:

- %LET does not allow leading or trailing blanks - for them use quotes or the %STR function.
- Quotes remain part of the macro variable value. Note that blanks can be included and other special characters too - see Station8.
- Macro variable references can themselves be used within %LET statements.
- Macro variable names conform to usual SAS naming conventions. Note that although WEALDSTONE is 10 characters long, the Log reports WEALDSTON - 9 Characters. Once the limit of 8 characters is exceeded SAS reports the error.
- %STR() hides the meaning of blanks and special characters, but not & and %. A macro variable reference will resolve within the function.
- %NRSTR() hides the meaning of the % and& as well as blanks and other special characters. A macro variable reference within the argument to the function will NOT resolve.

We have seen that the Global Symbol Table is built at SAS invocation time. All the %LETs in the example above will also be written to this table, and after they have executed the table will contain the following:

Definition of a Macro

As the module title implies, a macro is simply 'a bundle of code'. In its simplest form, the regular job you submit to backup your data sets or produce your graphics can all be bundled up into a macro and then 'invoked' using one word - the name of the macro. This is known as making a macro 'call'.

But as you would expect, the macro facility can do far more than this. For example, we shall see how to pass values to the macro, and, in Module 5, how to take decisions within the macro.

All this simply leads to the inclusion of text (normally ordinary SAS code - DATA and PROC steps) on to the top of the input stack, to be submitted to SAS in the normal way.

Example

Consider a regular job to age a series of time-related data sets; there are four data sets in the group. A new data set forms the first, newest member, the former first data set becomes the second and so on:

PROGRAM EDITOR

```

data mylib.new;
...
... data step statements ...
...
proc datasets lib=mylib;
age new first second third fourth;
run;

```

new	is renamed	first
first	is renamed	second
second	is renamed	third
third	is renamed	fourth
fourth	is deleted	

How can we bundle this code up into a macro?

Macro definition

- A macro must be defined first before it can be called.
- Macro definitions start with the %MACRO statement which defines the name of the macro...
- ... and the definition is deemed to continue until the %MEND statement. Including the macro name is optional (but if it is used must correspond with the name of the macro):

```

%macro macroname;
.
macro programming statements;
.
%mend macroname;

```

The above example thus becomes:

PROGRAM EDITOR

```

%macro age; /*definition of macro called age*/ proc
datasets lib=mylib;
age new first second third fourth;
run;
%mend age; /*completion of macro definition*/

```

N.B. The %MEND is critical. The macro processor takes control when a %MACRO statement is seen. Should the %MEND be

missing, all the input stream is regarded as being part of the open macro definition. There are occasions when all the submitted code is seen to be written to the log and nothing else - the code appears to be disappearing into a black hole! Upon such occasions, check for the absence of a %MEND statement.

What can a macro contain?

- Data and Proc step code
- Macro programming statements and functions

We shall see these in Module M5. Some macro programming statements must be within the bounds of a macro definition, others are totally global.

- Macro variable references
i.e. things starting with the &
- Other macro calls and definitions
i.e. things starting with the %

In the example we have just seen, the name of the data sets are fixed. We shall see how to make these variable shortly.

What can a macro be called?

- Any valid SAS name (CMS users are limited to a maximum of 7 characters)
- Anything other than one of the following reserved words:

Reserved words in the macro facility (Release 6.06 and higher):

ABEND	GOTO	QUOTE
ABORT	IF	QUPCASE
ACT	INC	RESOLVE
ACTIVATE	INCLUDE	RETURN
BQUOTE	INDEX	RUN
BY	INFILE	SAVE
CLEAR	INPUT	SCAN
CLOSE	KEYDEF	STOP
CMS	LENGTH	STR
COMANDR	LET	SUBSTR
COPY	LIST	SUPERQ
DEACT	LISTM	SYSEXEC
DEL	LOCAL	SYSGET
DELETE	MACRO	SYSRPUT
DISPLAY	MEND	THEN
DMIDSPLY	METASYM	TO
DMISPLIT	NRBQUOTE	TSO
DO	NRQUOTE	UNQUOTE
EDIT	NRSTR	UNSTR
ELSE	ON	UNTIL
END	OPEN	UPCASE
EVAL	PAUSE	WHILE
FILE	PUT	WINDOW
GLOB	QSCAN	
GO	QSUBSTR	

An attempt to call a macro by one of the above reserved names will result in a warning message; the macro will neither be compiled or available for use.

The compiled macro

- A compiled macro is an entry in a utility catalog in the WORK library
- the system does not support the renaming or copying of entries of member type of macro

The once-defined macro is stored in the WORK data library in compiled form. In versions prior to 6.03 it was stored as a special type of data set; in version 6.03 and above, it is stored in the WORK.SASMACR catalog with an entry type of MACRO. As it is held in 'compiled' form, it is, of course, not browsable or editable.

The macro could also be part of an autocall library. This topic will be discussed at length in a later module.

The Macro Call

Once a macro has been defined it can be called anywhere in a SAS job. The call is simply the name of the macro preceded by the % sign:

PROGRAM EDITOR

```
%macro age; /*definition of macro called age*/
proc datasets lib=mylib;
    age new first second third fourth;
run;
%mend age; /*completion of macro definition*/
data mylib.new;
    set...;
    if....;
run;
%age                /*the macro call*/
```

Notice that the macro call, %age, does not include a semi-colon. There is no need here as a semi-colon has been generated by the macro call; the code within the definition is complete, so no extra semi-colon is required.

The macro call sees the execution phase of the macro, whereupon the macro processor executes the macro in sequential form, placing the resulting open code (i.e. simple DATA and PROC steps) upon the input stack.

Passing Parameters

In the %age example the data set names were fixed. How could a macro be written such that the procedure is invoked with any names for the library and data sets involved? i.e. to generate:

PROGRAM EDITOR

```
proc datasets lib=mylib;
age new first second third fourth;
run;
```

at the first invocation and to generate

PROGRAM EDITOR

```
proc datasets lib=newlib;
age next ds_0 ds_1 ds_2;
run;
```

at the next?

The way to do this is to pass parameters to the macro call.

To use this method, the macro must first be defined as requiring parameters in the call. There are two ways of doing this:

Positional Parameters

PROGRAM EDITOR

```
data new; x=5; run;
data first ; x=1; run;
data second; x=2; run;
data third ; x=3; run;
data fourth; x=4; run;

%macro age(library,newds,ds1,ds2,ds3,ds4);
proc datasets lib=&library;
  age &newds &ds1 &ds2 &ds3 &ds4;
run;
quit;
%mend;

%age(work,new,first,second,third,fourth)
```

Here the macro age has been defined with six positional parameters to take the variation in library and data sets - in order.

The macro is invoked by:

```
%age(sasdata,latest,prod1,prod2,prod3,prod4)
```

to generate:

PROGRAM EDITOR

```
proc datasets lib=sasdata;
age latest prod1 prod2 prod3 prod4;
run;
```

What would the call

```
%age(prod1,prod2,prod3,prod4,sasdata,latest)
```

generate?

Keyword Parameters

This method does exactly the same job as defining a macro with positional parameters, except:

- it gets over the requirement to define and pass parameters in the same order
- it allows default values to be attached to the parameter

PROGRAM EDITOR

```
data new; x=5; run;
data first ; x=1; run;
data second; x=2; run;
data third ; x=3; run;
data fourth; x=4; run;

%macro
age(library=work,newds=new,ds1=,ds2=,ds3=,ds4=);
proc datasets lib=&library;
  age &newds &ds1 &ds2 &ds3 &ds4;
run;
quit;
%mend;

%age(ds1=first,ds2=second,ds3=third,ds4=fourth)
```

The call

```
%age(ds1=first,ds2=second,ds3=third,ds4=fourth)
```

generates...

PROGRAM EDITOR

```
proc datasets lib=mylib;
age work1 first second third fourth;
run;
```

the definition of the macro providing default values for &library and &newds.

The call

```
%age(ds2=april,ds4=june,ds1=march,ds3=may,library=y
earlib)
```

generates

PROGRAM EDITOR

```
proc datasets lib=yearlib;
age work1 march april may june;
run;
```

Where the macro is defined with parameters the parentheses MUST be used. For example, where all the parameters are given default values, the minimum invocation is:

```
%age()
```

Null values

With positional parameters, null values can be passed by using a comma as a 'placeholder':

PROGRAM EDITOR

```
%macro rr(datads,setds,condval);
  data &datads;
    set &setds;
    if category="&condval";
  run;
%mend rr;

%rr(work9,saved.epidemic,E)
%rr(,saved.epidemic,E)
```

```
%rr(,work3,F)
```

will generate:

PROGRAM EDITOR

```
data;
set work3;
if key="F";
run;
```

and SAS will choose the name of the temporary output data set.

```
%rr(.,M)
```

will generate:

PROGRAM EDITOR

```
data;
set;
```

```
if key="M";  
run;
```

and SAS will choose the name of the temporary output data set and use the last updated data set as input.

With keyword parameters, the parameter is simply omitted.

Combination of Positional and Keyword parameters

If the methods of positional and keyword parameters are mixed, the positional parameters must come first.

PROGRAM EDITOR

```
%macro tt(proc,dataset=_last_);  
  proc &proc data=&dataset;  
  run;  
%mend tt;
```

The call...

```
%tt(print,dataset=first)
```

...generates

PROGRAM EDITOR

```
proc print data=first;  
run;
```

Variable Numbers of Parameters

Sometimes you may want to write a macro to contain variable numbers of parameters. For example, the %age macro in its forms defined so far can only age 5 data sets; what if we wanted to write a utility macro so we could age any number of data sets. A way around this is to use the PARMBUFF option.

- Define the macro in the normal way except for the /PARMBUFF option.

```
%macro age/parmbuff;
```

macro programming statements

```
%mend age;
```

Here, all supplied parameters, including any special characters used are assigned to the automatic local macro variable SYSPBUFF which is then manipulated in the macro by macro programming statements.

The call

```
%age(library=mylib,new,gdg_0,gdg_1,gdg_2)
```

gives a value to &syspbuff of library=mylib,new,gdg_0,gdg_1,gdg_2

Parameters may also be included in the definition

```
%macro age(posparm)/parmbuff;
```

macro programming statements

```
%mend age;
```

In the above example, a different number of parameters can be supplied as long as there is at least one.

The call

```
%age(mylib,new,gdg_0,gdg_1,gdg_2)
```

gives a value to &syspbuff of
mylib,new,gdg_0,gdg_1,gdg_2

and

&posparm the value mylib.