The *macro facility* is a tool for extending and customizing SAS and for reducing the

amount of text you must enter to do common tasks.

The macro facility has two components:

* *macro processor* is the portion of SAS that does the work
* *macro language* is the syntax that you use to communicate with the macro processor.

macro triggers: (& and %)

&*name* refers to a macro variable. “Replacing Text Strings Using Macro

Variables” on page 4 explains how to create a macro variable. The

form &*name* is called a macro variable reference.

%*name* refers to a macro. “Generating SAS Code Using Macros” on page 5

explains how to create a macro. The form %*name* is called a macro

call..

title "Data for &city";

The macro processor resolves the reference to the macro variable CITY:

title "Data for New Orleans";

In quoted strings in open

code, the macro processor resolves macro variable references within double quotation

marks but not within single quotation marks.

open code (outside a macro definition)

%LET statement in open code (outside a macro definition) creates a global macro

variable that is available for use anywhere.

Defining Macros

Macros enable you to substitute text in a program and to do many other things. A

SAS program can contain any number of macros, and you can invoke a macro any

number of times in a single program.

%MACRO *macro-name*;

*macro definition*

%MEND *macro-name*;

To call (or *invoke*) a macro, precede the name of the macro with a percent sign (%):

%*macro-name*

Passing Information into a Macro Using Parameters

%macro plot(yvar= ,xvar= );

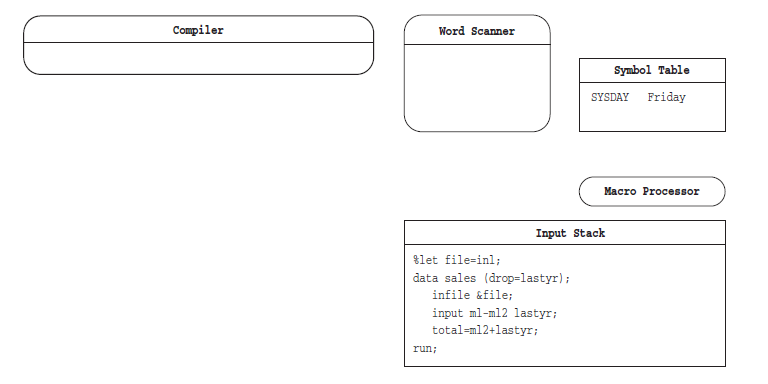
proc plot;

plot &yvar\*&xvar;

run;

%mend plot;

Basic Process:



Macro variables defined by macro programmers are called *user-defined macro*

*variables*. Those defined by the macro processor are called *automatic macro variables*.

Use %PUT \_AUTOMATIC\_ to view all available automatic macro variables.

the macro processor treats alphabetic characters, digits, and symbols

(except & and %) as characters.

**%let totalstr=100+200;**

**%let num=%eval(100+200);**

**%let status=%wait;**

The macro processor stores the text produced by the macro WAIT

as the value of STATUS.

**%let status=%nrstr(%wait);**

The macro processor stores **%wait** as the value of STATUS.

the %NRSTR function to mask the percent sign

macro quoting function %STR or %NRSTR around the value. This action masks

the blanks or special characters so that the macro processor interprets them as

text.

**%let state=%str( North Carolina);**

**%let town=%str(Taylor%’s Pond);**

**%let store=%nrstr(Smith&Jones);**

**%let plotit=%str(**

**proc plot;**

**plot income\*age;**

**run;);**

To resolve a macro variable reference that occurs within a literal string, enclose the string in double

quotation marks.

%let dsn=Newdata;

title1 "Contents of Data Set &dsn";

title2 ’Contents of Data Set &dsn’;

However, you can use a period ( . ) as a delimiter for a macro variable reference

%let name=sales;

data &name.1 &name.2;

SAS now sees this statement:

DATA SALES1 SALES2;

The system option SYMBOLGEN displays the resolution of macro variables.

Referencing Macro Variables Indirectly

%put &city&n; /\* incorrect \*/

%put &&city&n; /\* correct \*/

When you use an indirect macro variable reference, you must force the macro

processor to scan the macro variable reference more than once and resolve the desired

reference on the second, or later, scan.

Macro Functions

%let address=123 maple avenue;

%let frstword=%scan(&address,1);

the %SCAN function to search the source

(first argument) and retrieve the first word (second argument).

Defining and Calling Macros

When you submit a macro definition, the macro processor compiles the definition and

produces a member in the session catalog. The member consists of compiled macro

program statements and text.

Examples of text items include:

􀀀 macro variable references

􀀀 nested macro calls

􀀀 macro functions, except %STR and %NRSTR

􀀀 arithmetic and logical macro expressions

􀀀 text to be written by %PUT statements

􀀀 field definitions in %WINDOW statements

􀀀 model text for SAS statements and SAS windowing environment commands

Scopes of Macro Variables

A macro variable’s scope determines how it is

assigned values and how the macro processor resolves references to it.

use the %SYMEXIST function to indicate whether a macro variable exists.

Global macro variables include the following:

* macro variables created outside of any macro.
* macro variables created in %GLOBAL statements.
* most macro variables created by the CALL SYMPUT routine

%put \_user\_;

\_all\_

\_global\_

\_local\_

%macro env1(param1);

data \_null\_;

x = ’a token’;

call symput(’myvar1’,x);

run;

%put \*\* Inside the macro: \*\*;

%put \_user\_;

%mend env1;

%let d=%eval(10.0+20.0); /\*INCORRECT\*/

%let a=%sysevalf(10.0\*3.0);

Macro Quoting

Macro quoting functions resolve these ambiguities by masking the significance of

special characters so that the macro processor does not misinterpret them.

To display information about compiled macros when you invoke them, use the SAS

system options MLOGIC, MPRINT, and SYMBOLGEN. When you specify the SAS

system option MLOGIC, the libref and date of compilation of a stored compiled macro

are written to the log along with the usual information displayed during macro

execution.

%macro temp;

data one;

%do i=1 %to 3;

x&i=&i;

%end;

run;

%mend temp;

%temp

SYMGET function returns the value of a macro variable to the DATA step during DATA

step execution.

SYMPUT routine assigns a value produced in a DATA step to a macro variable.

INTO assigns the result of a calculation or the value of a data column.

data team2;

input position : $12. player $12.;

call symput(’POS’||left(\_n\_), position);

datalines;

shortstp Ann

pitcher Tom

frstbase Bill

;

If *macro-variable* does not exist, SYMPUT creates it. SYMPUT makes a macro variable

assignment when the program executes.

Comparisons

􀀀 SYMPUT assigns values produced in a DATA step to macro variables during

program execution, but the SYMGET function returns values of macro variables to

the program during program execution.

Example 1: Creating Macro Variables and Assigning Them Values from a Data Set

data dusty;

input dept $ name $ salary @@;

datalines;

bedding Watlee 18000 bedding Ives 16000

bedding Parker 9000 bedding George 8000

bedding Joiner 8000 carpet Keller 20000

carpet Ray 12000 carpet Jones 9000

gifts Johnston 8000 gifts Matthew 19000

kitchen White 8000 kitchen Banks 14000

kitchen Marks 9000 kitchen Cannon 15000

tv Jones 9000 tv Smith 8000

tv Rogers 15000 tv Morse 16000

;

proc means noprint;

class dept;

var salary;

output out=stats sum=s\_sal;

run;

data \_null\_;

set stats;

if \_n\_=1 then call symput(’s\_tot’,trim(left(s\_sal)));

else call symput(’s’||dept,trim(left(s\_sal)));

run;

%put \_user\_;

When this program executes, this list of variables is written to the SAS log:

GLOBAL SCARPET 41000

GLOBAL SKITCHEN 46000

GLOBAL STV 48000

GLOBAL SGIFTS 27000

GLOBAL SBEDDING 59000

GLOBAL S\_TOT 221000

%put \_user\_;

SYMGET Function

**Returns the value of a macro variable to the DATA step during DATA step execution.**

Example 1: Retrieving Variable Values Previously Assigned from a Data Set

data dusty;

input dept $ name $ salary @@;

datalines;

bedding Watlee 18000 bedding Ives 16000

bedding Parker 9000 bedding George 8000

bedding Joiner 8000 carpet Keller 20000

carpet Ray 12000 carpet Jones 9000

gifts Johnston 8000 gifts Matthew 19000

kitchen White 8000 kitchen Banks 14000

kitchen Marks 9000 kitchen Cannon 15000

tv Jones 9000 tv Smith 8000

tv Rogers 15000 tv Morse 16000

;

proc means noprint;

class dept;

var salary;

output out=stats sum=s\_sal;

run;

proc print data=stats;

var dept s\_sal;

title "Summary of Salary Information";

title2 "For Dusty Department Store";

run;

data \_null\_;

set stats;

if \_n\_=1 then call symput(’s\_tot’,s\_sal);

else call symput(’s’||dept,s\_sal);

run;

data new;

set dusty;

pctdept=(salary/symget(’s’||dept))\*100;

pcttot=(salary/&s\_tot)\*100;

run;

proc print data=new split="\*";

label dept ="Department"

name ="Employee"

pctdept="Percent of \*Department\* Salary"

pcttot ="Percent of \* Store \* Salary";

format pctdept pcttot 4.1;

title "Salary Profiles for Employees";

title2 "of Dusty Department Store";

run;

INTO Clause

**Assigns values produced by PROC SQL to macro variables.**

select style, sqfeet

into :type, :size

from sasuser.houses;

specifies a numbered list of macro variables. Leading and trailing blanks are

trimmed from values before they are stored in macro variables.

select distinct style

into :types separated by ’,’

from sasuser.houses;