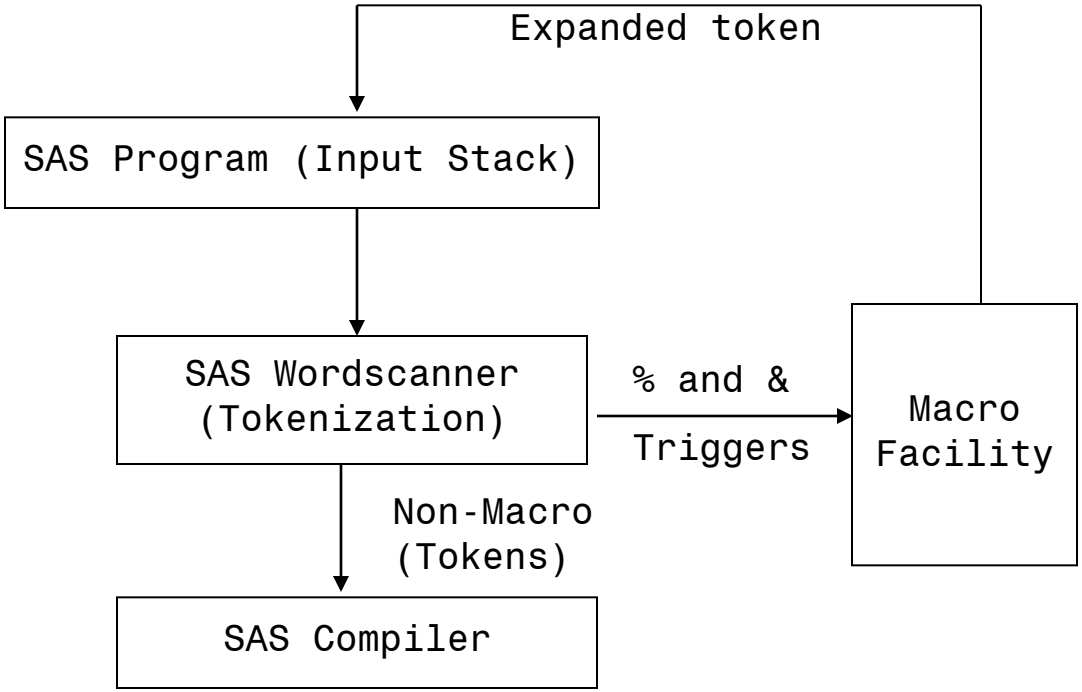
An Introduction to SAS® Macros



SYSTEMS SEMINAR CONSULTANTS, INC.

Steven First, President
2997 Yarmouth Greenway Drive, Madison, WI 53711
Phone: (608) 278-9964, Web: www.sys-seminar.com

Questions, Comments

- Technical Difficulties: Call 1-800-263-6317
- We have several hundred attendees, so we won't be able to answer questions live.
- Please email questions or comments to train@sys-seminar.com.
- A copy of the presentation and a recording of the webinar will be emailed out to attendees. This can be shared with people who did not attend.

Systems Seminar Consultants, Inc.

Consulting, Training, and Support

- SAS
- SQL
- ETL
- Reporting Systems
- Business Analytics
- Data modeling
- Automating Processes
- “Big Data”

Apply good IT and systems practices to real life business applications.

Work with Marketing, Finance, Retail, Insurance, Utilities, Manufacturing, Healthcare, Government organizations.

Presenter – Teresa Schudrowitz

- Senior Consultant, Project Manager, and SAS Trainer
- Data conversion, automated reporting systems, production support, application development, planning and analysis, and project planning.
- B.A. in Management Information Systems, B.A in Accounting, and M.S.in Management Information Systems.
- Presenter at at many local user group meetings, regional user group meetings, PharmaSUG, and SAS Global Forum.

SAS Macro Overview

Macros construct input for the SAS compiler.

Functions of the SAS macro processor:

- pass symbolic values between SAS statements and steps
- establish default symbolic values
- conditionally execute SAS steps
- invoke very long, complex code in a quick, short way

Notes:

- The MACRO PROCESSOR is the SAS system module that processes macros.
- The MACRO LANGUAGE is the means of communicating with the processor.

Traditional SAS Programming

Without macros what are some things you *cannot* easily do?

- substitute text in statements like TITLEs
- communicate across SAS steps
- establish default values
- conditionally execute SAS steps
- hide complex code that can be invoked easily

Traditional SAS Programming (continued)

Without macros, SAS programs are a series of DATA and PROC steps.

1. The program is scanned one statement at a time looking for the beginning of a step (step boundary).
2. When the beginning of a step is found, all statements in the step are **compiled**.
3. When the end of the step is found (the next step boundary), the previous step **executes**.

Notes:

- Some macro features like %LET, and & are processed at word scan time.
- Some statements such as IF, CALL SYMPUT and SYMGET are processed at data step execution time.

Traditional SAS Programming (continued)

Step boundaries are the SAS keywords:

DATA	ENDSAS
PROC	LINES
CARDS	LINES4
CARDS4	PARMCARDS
DATALINES	QUIT
DATALINES4	RUN

Notes:

- Each step is compiled and executed independently.
- All SAS jobs have a step boundary at the beginning.
- Batch and non-interactive jobs have an implied ENDSAS at the end of the SYSIN file.
- RUN and QUIT have special considerations.

SAS Steps

Step boundaries control the compilation and execution of SAS programs.

```
                                        <------ Step, start compile
Data saleexps;
 Infile rawin;
 Input  name $1-10 division $12
        years 15-16 sales 19-25
        expense 27-34;
                                        <------ Step, exec prev,
                                               start compile
Proc print data=saleexps;
                                        <------ Step, exec prev,
                                               start compile
Proc means data=saleexps;
 Var sales expense;
                                        <------ EOF (batch), exec
                                               prev, ENDSAS.
```

SAS Steps (continued)

Notes:

- Interactive jobs have no end-of-file, so the last step will compile but not execute.

The RUN Statement

RUN acts as an explicit step boundary in most PROCs.

```
                                        <------ Step, start compile
Data saleexps;
  Infile rawin;
  Input name $1-10 division $12
        years 15-16 sales 19-25
        expense 27-34;
  Run;                                  <------ Step end, exec prev

Proc print data=saleexps;               <------ Step, start compile
  Run;                                  <------ Step end, exec prev
Proc means data=saleexps;               <------ Step, start compile
  Var sales expense;
  Run;                                  <------ Step end, exec prev
```

Notes:

- The use of RUN after each step is HIGHLY recommended.

RUN-Group Processing

Many SAS PROCs use RUN to submit groups of statements without terminating the PROC.

```
Proc plot data=saleexps;          <---- Step, start compile
  Plot sales * expense;
Run;                              <---- Execute
  Plot years * expense;
Run;                              <---- Execute
  Plot sales * expense=division;
Run;                              <---- Execute
Proc datasets;                    <---- Step, start compile
  Exchange x=y;
Run;                              <---- Execute
  Delete a;
Quit;                             <---- Step, execute
```

RUN-Group Processing (continued)

Notes:

- QUIT is only used to terminate RUN-group PROCs. It is NOT a general boundary statement.
- If the last PROC submitted supports RUN-group processing, the display will say 'PROC *procname* running'.

SAS Global Statements

Global statements are executed immediately.

```
Filename rawin 'saleexps.dat';     ←— compile and execute
Data saleexps;                     ←— Step, start compile
  Infile rawin;
  Input name $1-10 division $12
        years 15-16 sales 19-25
        expense 27-34;
  Run;                             ←— Step end, exec prev

Proc print data=saleexps;          ←— Step, start compile
Options ls=80 nodate;              ←— compile and execute
Title 'sample title';              ←— compile and execute
  Run;                             ←— Step end, exec prev
Options ls=132 nodate;             ←— compile and execute
Proc means data=saleexps;          ←— Step, start compile
  Var sales expense;
  Run;                             ←— Step end, exec prev
```

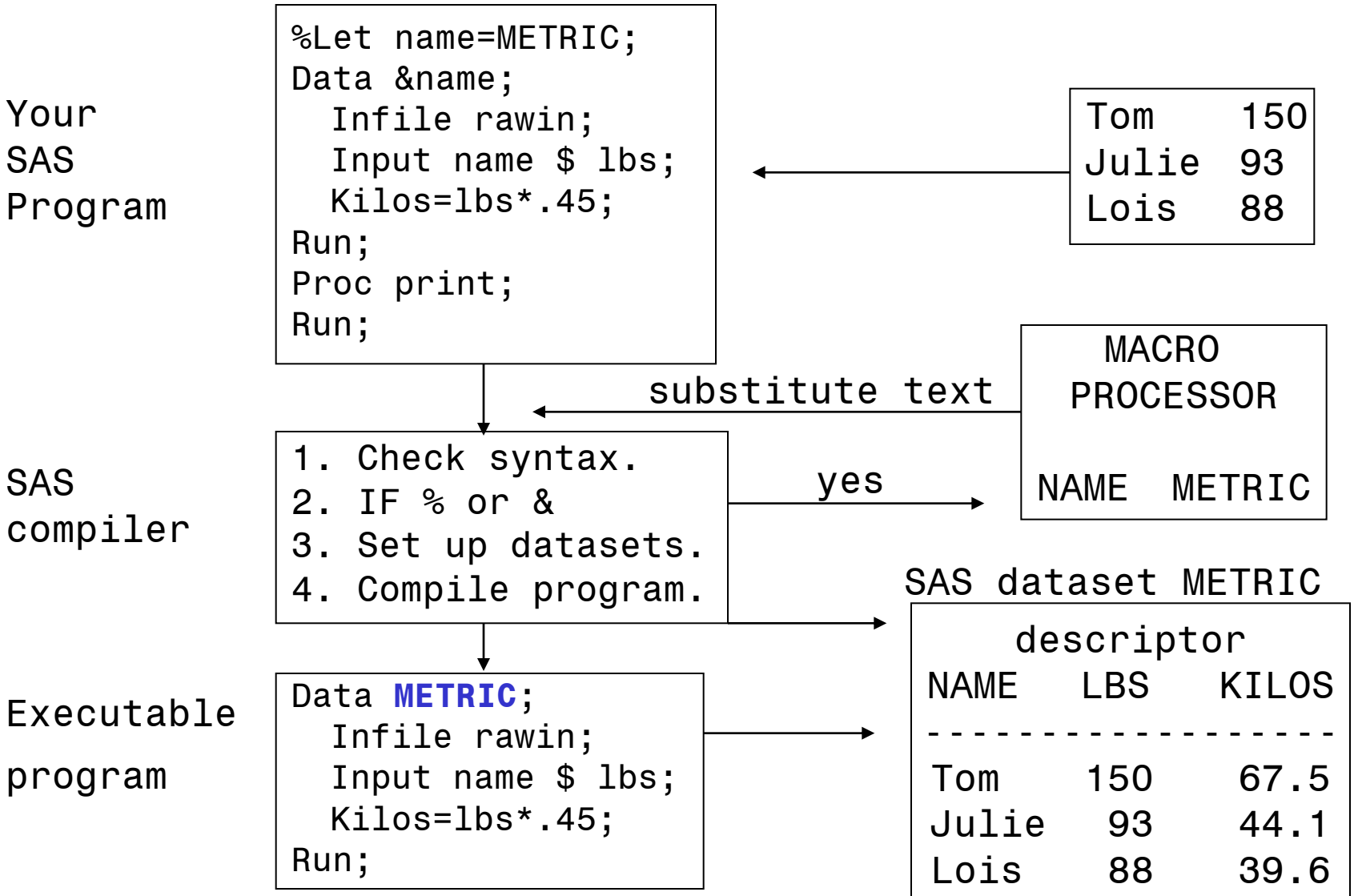
SAS Global Statements (continued)

Notes:

- RUN ensures that global statements affect the correct step.

Macro Processor Flow

Macro statements are given to the macro processor BEFORE the compiler.



The SAS Macro Language

A second SAS programming language for string manipulation.

Characteristics:

- Strings are sequences of characters.
- All input to the macro language is a string.
- Usually strings are SAS code, but they do not have to be.
- The macro processor manipulates strings and may send them back for scanning.

Macro Language Components

The macro language has several kinds of components.

Macro variables:

- are used to store and manipulate character strings
- follow SAS naming rules
- are NOT the same as DATA step variables
- are stored in memory in a macro symbol table

Macro statements:

- begin with a % and a macro keyword and end with a semicolon (;)
- assign values, substitute values, and change macro variables
- can branch or generate SAS statements conditionally

Macro Language Components (continued)

Macro functions:

- help process and evaluate text and macro variables
- have some capabilities of SAS data step functions
- have some unique capabilities

Macro expressions:

- are sequences of text linked with operators and parentheses
- are needed by some macro functions and programming statements

Macro constants:

- are treated as character strings
- can contain literals, variable names, numbers, dataset names, SAS statements

A Macro Problem

You reference a SAS data set name several times in a SAS job.

```
DATA PAYROLL;
  INPUT EMP$ RATE;
  DATALINES;
  TOM 10
  JIM 10
  ;
  PROC PRINT DATA=PAYROLL;
    TITLE "PRINT OF DATASET PAYROLL";
  RUN;
```

You would like to:

- be able to change the name quickly in only one place
- have the data set name appear in a title

Solution: Use a macro variable.

Macro Variables

- You can define macro variables with %LET.
- You refer to the variables later with &*variable*.
- Macro will substitute *value* for all occurrences of &*variable.*

Syntax:

%LET *variable=value*;

Macro Variables (continued)

Example:

```
%LET NAME=PAYROLL;
DATA &NAME;
  INPUT EMP$ RATE;
  DATALINES;
TOM 10
JIM 10
;
PROC PRINT DATA=&NAME;
  TITLE "PRINT OF DATASET &NAME";
RUN;
```

Macro Variables (continued)

Resolves to:

```
DATA PAYROLL;
  INPUT EMP$ RATE;
  DATALINES;
TOM 10
JIM 10

;
PROC PRINT DATA=PAYROLL;
  TITLE "PRINT OF DATASET PAYROLL";
RUN;
```

Notes:

- Macro variables are not resolved within single quotes.

Assigning a New Value

Use another %LET to assign a different value.

```
%LET NAME=NEWPAY;
DATA &NAME;
  INPUT EMP$ RATE;
  DATALINES;
TOM 10
JIM 10
;
PROC PRINT DATA=&NAME;
  TITLE "PRINT OF DATASET &NAME";
RUN;
```

Assigning a New Value (continued)

Resolves to:

```
DATA NEWPAY;
  INPUT EMP$ RATE;
  DATALINES;
TOM 10
JIM 10
;
PROC PRINT DATA=NEWPAY;
  TITLE "PRINT OF DATASET NEWPAY";
RUN;
```

Assigning SAS Statements to Macro Variables

%STR allows values to contain a semicolon(;) etc.

```
%LET NAME=NEWPAY;
%LET CHART=%STR(PROC CHART;VBAR EMP;RUN;);
DATA &NAME;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
&CHART
PROC PRINT DATA=&NAME;
 TITLE "PRINT OF DATASET &NAME";
RUN;
```

Assigning SAS Statements to Macro Variables (continued)

Resolves to:

```
DATA NEWPAY;
  INPUT EMP$ RATE;
  DATALINES;
TOM 10
JIM 10
;
PROC CHART;VBAR EMP;RUN;
PROC PRINT DATA=NEWPAY;
  TITLE "PRINT OF DATASET NEWPAY";
RUN;
```

Nesting of Macro Variables

Macro variables can contain other macro variables.

```
%LET NAME=NEWPAY;
%LET CHART=%STR(PROC CHART DATA=&NAME;VBAR EMP;RUN;);
DATA &NAME;
  INPUT EMP$ RATE;
  DATALINES;
TOM 10
JIM 10
;
&CHART
PROC PRINT DATA=&NAME;
  TITLE "PRINT OF DATASET &NAME";
RUN;
```

Nesting of Macro Variables (continued)

Resolves to:

```
DATA NEWPAY;
  INPUT EMP$ RATE;
  DATALINES;
TOM 10
JIM 10
;
PROC CHART DATA=NEWPAY;VBAR EMP;RUN;
PROC PRINT DATA=NEWPAY;
  TITLE "PRINT OF DATASET NEWPAY";
RUN;
```

Displaying Macro Variables

%PUT displays macro variables in the log at compile time.

Syntax:

%PUT *text macrovariables;*

Displaying Macro Variables (continued)

Example:

```
%LET NAME=NEWPAY;
%LET NAME2=OLDPAY;
DATA &NAME;
  INPUT EMP$ RATE;
  DATALINES;
TOM 10
JIM 10
;
RUN;
%PUT ***** NAME=&NAME NAME2=&NAME2 *****;
PROC PRINT DATA=&NAME;
  TITLE "PRINT OF DATASET &NAME";
RUN;
```

Partial SAS Log:

```
***** NAME=NEWPAY NAME2=OLDPAY *****
```

What is a Macro?

Stored text that can be inserted anywhere in a SAS program and expanded.

Macros can include:

- constants such as literals, variables, names, statements
- assignments to macro variables
- macro programming statements
- macro language functions
- invocations of other functions
- nested macro definitions

Defining and Using Macros

- %MACRO and %MEND define macros.
- *%macroname* will invoke it later.

Example: Define a macro to run PROC CHART and later invoke it.

Defining and Using Macros (continued)

```
%MACRO CHART;
  PROC CHART DATA=&NAME;
   VBAR EMP;
RUN;
%MEND;

%LET NAME=NEWPAY;
DATA &NAME;
  INPUT EMP$ RATE;
  DATALINES;
TOM 10
JIM 10
;
RUN;
%CHART
PROC PRINT DATA=&NAME;
  TITLE "PRINT OF DATASET &NAME";
RUN;
```

Defining and Using Macros (continued)

Resolves to:

```
DATA NEWPAY;
  INPUT EMP$ RATE;
  DATALINES;
TOM 10
JIM 10
;
RUN;
PROC CHART DATA=NEWPAY;
  VBAR EMP;
RUN;
PROC PRINT DATA=NEWPAY;
  TITLE "PRINT OF DATASET NEWPAY";
RUN;
```

Positional Macro Parameters

Macro parameters are defined in order after the macro name.

```
%MACRO CHART(NAME,BARVAR);
   PROC CHART DATA=&NAME;
     VBAR &BARVAR;
   RUN;
%MEND;

%CHART(PAYROLL,EMP)
```

Resolves to:

```
PROC CHART DATA=PAYROLL;
   VBAR EMP;
RUN;
```

Positioned Macro Parameters (continued)

Notes:

- Keyword parameters are also allowed.
- Keyword parameters can give default values.

Nested Macros

Macros can call other macros.

```
%MACRO CHART(NAME,BARVAR);
  PROC CHART DATA=&NAME;
   VBAR &BARVAR;
  RUN;
%MEND;

%MACRO PTCHART(NAME,BARVAR);
%CHART(&NAME,&BARVAR)
  PROC PRINT DATA=&NAME;
   TITLE "PRINT OF DATASET &NAME";
  RUN;
%MEND;

%PTCHART(PAYROLL,EMP)
```

Nested Macros (continued)

Resolves to:

```
PROC CHART DATA=PAYROLL;
  VBAR EMP;
RUN;
PROC PRINT DATA=PAYROLL;
  TITLE "PRINT OF DATASET PAYROLL";
RUN;
```

Conditional Macro Compilation

%IF can conditionally pass code to the compiler.

Example: Run PROC PRINT only if PRTCH=YES.

```
%MACRO PTCHT(PRTCH,NAME,BARVAR);
  %IF &PRTCH=YES %THEN PROC PRINT DATA=&NAME;
  ;
  PROC CHART DATA=&NAME;
   VBAR &BARVAR;
  RUN;
%MEND;

%PTCHT(YES,PAYROLL,EMP)
```

Conditional Macro Compilation (continued)

Resolves to:

```
PROC PRINT DATA=PAYROLL;

PROC CHART DATA=PAYROLL;
  VBAR EMP;
RUN;
```

Conditional Macro Compilation (continued)

%DO allows more than one statement to be conditionally compiled.

Example: Submit as before, but include titles.

```
%MACRO PTCHT(PRTCH,NAME,BARVAR);
  %IF &PRTCH=YES %THEN
    %DO;
      PROC PRINT DATA=&NAME;
      TITLE "PRINT OF DATASET &NAME";
      RUN;
    %END;
 PROC CHART DATA=&NAME;
 VBAR &BARVAR;
 RUN;
%MEND;

%PTCHT(YES,PAYROLL,EMP)
```

Conditional Macro Compilation (continued)

Resolves to:

```
PROC PRINT DATA=PAYROLL;
  TITLE "PRINT OF DATASET PAYROLL";
RUN;
PROC CHART DATA=PAYROLL;
  VBAR EMP;
RUN;
```

Iterative Macro Invocation

%DO can also vary a value.

Example: Run PROC PRINT &PRTNUM times.

```
%MACRO PRTMAC(PRTNUM,NAME);
 %DO I= 1 %TO &PRTNUM;
   PROC PRINT DATA=&NAME&I;
     TITLE "PRINT OF DATASET &NAME&I";
   RUN;
 %END;
%MEND;

%PRTMAC(4,PAYROLL)
```

Iterative Macro Invocation (continued)

Resolves to:

```
PROC PRINT DATA=PAYROLL1;
  TITLE "PRINT OF DATASET PAYROLL1";
RUN;
PROC PRINT DATA=PAYROLL2;
  TITLE "PRINT OF DATASET PAYROLL2";
RUN;
PROC PRINT DATA=PAYROLL3;
  TITLE "PRINT OF DATASET PAYROLL3";
RUN;
PROC PRINT DATA=PAYROLL4;
  TITLE "PRINT OF DATASET PAYROLL4";
RUN;
```

SAS DATA Step Interfaces

SYMGET, SYMPUT, and macro variables can transfer values between SAS steps.

Example: Display the number of observations in a dataset in a title.

```
%MACRO OBSCOUNT(NAME);
  DATA _NULL_;
    SET &NAME NOBS=OBSOUT;
    CALL SYMPUT('MOBSOUT',OBSOUT);
    STOP;
RUN;
PROC PRINT DATA=&NAME;
  TITLE "DATASET &NAME CONTAINS &MOBSOUT OBSERVATIONS";
RUN;
%MEND;

%OBSCOUNT(PAYROLL)
```

SAS DATA Step Interfaces

Resolves to:

```
DATA _NULL_;
  SET PAYROLL NOBS=OBSOUT;
  CALL SYMPUT('MOBSOUT',OBSOUT);
  STOP;
RUN;
PROC PRINT DATA=PAYROLL;
  TITLE  "DATASET PAYROLL CONTAINS 50 OBSERVATIONS";
RUN;
```

Notes:

- SYMGET returns macro variable values to the DATA step.

A SAS Macro Application

The following problem needs some help:

PERM.COUNTYDT

Obs	COUNTYNM	READING
1	ASHLAND	125
2	ASHLAND	611
3	BAYFIELD	101
4	BAYFIELD	101
5	BAYFIELD	222
6	WASHINGTON	143

- Each day you read a SAS dataset containing data from counties in Wisconsin.
- Anywhere between 1 and 72 counties might report that day.

A SAS Macro Application

Do the following:

1. Create a separate dataset for each reporting county.
2. Produce a separate PROC PRINT for each reporting county.
3. In the TITLE print the county name.
4. Reset the page number to 1 at the beginning of each report.
5. In a footnote print the number of observations processed for each county.

How do you do it?

The Solution

Solution: Write a SAS Macro. A multi-step macro can communicate between steps.

```
DATA _NULL_;
   SET PERM.COUNTYDT END=EOF;          /* READ SAS DATASET     */
   BY COUNTYNM;                        /* SORT SEQ             */
   IF FIRST.COUNTYNM THEN DO;          /* NEW COUNTY ?         */
      NUMCTY+1;                        /* ADD 1 TO NUMCTY      */
      CTYOBS=0;                        /* OBS PER COUNTY TO 0  */
      END;
    CTYOBS+1;                          /* ADD 1 OBSER FOR CTY  */
   IF LAST.COUNTYNM THEN DO;           /* EOF CTY,MAKE MAC VARS*/
     CALL SYMPUT('MCTY'!!LEFT(PUT(NUMCTY,3.)),COUNTYNM);
     CALL SYMPUT('MOBS'!!LEFT(PUT(NUMCTY,3.)),LEFT(CTYOBS));
     END;
   IF EOF THEN                         /* VERY LAST OBS ?      */
     CALL SYMPUT('MTOTCT',NUMCTY);     /* MAC VAR NO DIF CTYS  */
   RUN;
```

The Solution (continued)

```
%PUT *** MTOTCT=&MTOTCT;                  /* DISPLAY NO OF CTYS   */

%MACRO COUNTYMC;                          /* MACRO START          */
%DO I=1 %TO &MTOTCT;                      /* LOOP THRU ALL CTYS   */
 %PUT *** LOOP &I OF &MTOTCT;             /* DISPLAY PROGRESS     */
 PROC PRINT DATA=PERM.COUNTYDT;           /* PROC PRINT           */
   WHERE COUNTYNM="&&MCTY&I";             /* GENERATED WHERE      */
   OPTIONS PAGENO=1;                      /* RESET PAGENO         */
   TITLE "REPORT FOR COUNTY &&MCTY&I";
                                          /* TITLES & FOOTNOTES   */
   FOOTNOTE "TOTAL OBSERVATION COUNT WAS &&MOBS&I";
 RUN;
%END;                                     /* END OF %DO           */
%MEND COUNTYMC;                           /* END OF MACRO         */
%COUNTYMC                                 /* INVOKE MACRO         */
```

The Generated Code and Output

```
*** MTOTCT=3
*** LOOP 1 OF 3
    PROC PRINT DATA=PERM.COUNTYDT;
    WHERE COUNTYNM="ASHLAND";   OPTIONS PAGENO=1;
    TITLE "REPORT FOR COUNTY ASHLAND";
    FOOTNOTE "TOTAL OBSERVATION COUNT WAS 2";   RUN;
*** LOOP 2 OF 3
    PROC PRINT DATA=PERM.COUNTYDT;
    WHERE COUNTYNM="BAYFIELD";   OPTIONS PAGENO=1;
    TITLE "REPORT FOR COUNTY BAYFIELD";
    FOOTNOTE "TOTAL OBSERVATION COUNT WAS 3";   RUN;
*** LOOP 3 OF 3
    PROC PRINT DATA=PERM.COUNTYDT;
    WHERE COUNTYNM="WASHINGTON";   OPTIONS PAGENO=1;
    TITLE "REPORT FOR COUNTY WASHINGTON";
    FOOTNOTE "TOTAL OBSERVATION COUNT WAS 1";  RUN;
```

The Generated Code and Output (continued)

REPORT FOR COUNTY **ASHLAND** 1

OBS	COUNTYNM	READING
1	ASHLAND	125
2	ASHLAND	611

TOTAL OBSERVATION COUNT WAS **2**

REPORT FOR COUNTY **BAYFIELD** 1

OBS	COUNTYNM	READING
3	BAYFIELD	101
4	BAYFIELD	101
5	BAYFIELD	222

TOTAL OBSERVATION COUNT WAS **3**

REPORT FOR COUNTY **WASHINGTON** 1

OBS	COUNTYNM	READING
6	WASHINGTON	143

TOTAL OBSERVATION COUNT WAS **1**

A Solution Via PROC SQL

PROC SQL can create macro variables.

```
proc sql noprint;
  select  left(put(count(distinct countynm),3.))
           into:mtotct from countydt;
                                  /* no of unique ctys     */
  select  distinct countynm       /* each cty name         */
          into:mcty1-:mcty&mtotct from countydt;
  select  count(*)                /* obs per               */
          into:mobs1-:mobs&mtotct from countydt
  group by countynm;
quit;

%put *** mtotct=&mtotct           /* display no of ctys    */
     *** sqlobs=&sqlobs;          /* sql obs count         */
```

The Macro Is Unchanged

```
%macro countymc;                    /* macro start           */
%do i=1 %to &mtotct;                /* loop thru all ctys    */
 %put *** loop &i of &mtotct;       /* display progress      */
 proc print data=countydt;          /* proc print            */
  where countynm="&&mcty&i";        /* generated where       */
  options pageno=1;                 /* reset pageno          */
  title "report for county &&mcty&i";
                                    /* titles and footnotes  */
 footnote "total observation count was &&mobs&i";
 run;
%end;                               /* end of %do            */
%mend countymc;                     /* end of macro          */
%countymc                           /* invoke macro          */
```

Contact Us

SYSTEMS SEMINAR CONSULTANTS, INC.

SAS® Training, Consulting, & Placement Services

1-800-997-7081

608) 278-9964

train@sys-seminar.com

www.sys-seminar.com

2997 Yarmouth Greenway Drive • Madison, WI 53711

sas | SAS Alliance *Affiliate Member*