

SAS Macro Techniques for Generating Customized Reports

Jim DeFoor, General Dynamics – Fort Worth

OUTLINE

Introduction
Actions Performed by Program
Macro Techniques Used
Report Examples
Explanation of Macro Techniques
SAS Code Generated by Macro Code

INTRODUCTION

The Macro techniques shown in this paper allow one program to generate many different reports, each with its own unique numeric columns and logical page sorts. That capability permits users to report or analyze many different 'cuts' of their data with one program. Having only one program perform a multitude of reporting tasks reduces the development and maintenance activities of the supporting programmer. The techniques are used in a program at General Dynamics - Fort Worth (GDFW) that generates cost estimate reports of F-16 aircraft. The program has been used for every major proposal at GDFW during the last five years.

Three reports generated by this program are shown in this paper. Each was created by the unique SAS code generated by the one Macro program. That Macro code created the SAS code according to the program-fixed row variable of the report and the user's selection of the report's column and page-sort variables. It created individual column variables from each data value of the column variable selected by the user. It then used those column variables and the page sorts selected by the user to create the specific SORT, SUM, TRANSPOSE, and PUT operations necessary to organize and report the data in the structure desired by the user.

The three report examples all show man-hours of effort. Cost Element Code CEC is the fixed row variable of each report. It explains the organization contributing the man-hours of effort. The column variable of the reports are, in order, NRRC, PRRE, and YEAR. The data values of each one of those variables is the column heading of each report. For example, the column headings of the YEAR report are __1988, __1989, __1990, etc.

The SAS Code Generated by Macro Code section is organized in parallel subsections. The reader will find the SAS code under the same subheading as the Macro code which generated it. This should aid the reader in understanding the form and purpose of the Macro code in each subsection by allowing line-by-line comparisons of the created code versus the generating code.

Only the significant Macro code in the program is shown in this paper. That code is shown in the order of its use in the actual program. The complete program reads an external data set, assigns the data to cost groups, sorts and sums the data, transposes the data to the structure needed for the report, and then reports the data. The comments embedded in the Macro code explain the purpose and function of the following or adjoining lines of Macro code. The comments in the SAS code explain only how the Macro code was resolved.

The section that explains the Macro techniques used in the program is meant as a brief reference of those techniques. Complete explanations may be found in the *SAS Guide to Macro Processing, Version 6, Second Edition*. Page citations are provided.

ACTIONS PERFORMED BY PROGRAM

These actions are described in the section SAS Code Generated by Macro Code.

1. Collect the variables that will generate the columns and the logical page breaks of the report.
2. Sort and sum the data according to the column variable, the page-sort variables, and the program's fixed-row variable (CEC).
3. Build Macro variables from the data values of the selected column variable (NRRC, PRRE, or YEAR).
4. Create a Data Step for transposing the data (one that is more efficient than PROC TRANSPOSE). Generate an SAS variable for each of the Macro variables that represent the new column variables.
5. Establish a report-writing Data _Null_ Step that
 - A. Generates a new logical print page for each change in the data values of that last sort variable specified.
 - B. Builds a TOTAL variable of the new SAS column variables.
 - C. Writes the column data on to the report using PUT statements.
 - D. Prints the values of the page-sort variables below the titles and above the column headings of the report using PUT statements in the HEADER area of the Data _Null_ Step. Prints 'TOTAL' if no sort-variables were specified.
 - E. Writes the column headings on to the report using PUT statements in the HEADER area of the Data _Null_ Step.

MACRO TECHNIQUES USED IN PROGRAM

These are the Macro techniques used in the program. Several occur in more than one subsection of SAS Code Generated by Macro Code. For example, %LENGTH is used in subsections 4 and 5E; %EVAL, in Sections 4 and 5C.

- Assigning a Macro variable with a %LET statement
- Simple Macro variable references.
- Assigning a Macro variable with a CALL SYMPUT function
- Iterative %DO statements.
- %IF-%THEN conditional execution.
- %SUBSTR function.
- %LENGTH function.
- %EVAL function
- Double resolution of Macro variables.
- %DO %WHILE statements.

EXAMPLES OF REPORTS PRODUCED BY PROGRAM

REPORT GENERATED FROM COLUMN = NRRC

(Nonrecurring/Recurring)

08/27/90 15:07:49

General Dynamics

Fort Worth Divison – Fort Worth, Texas

STANDARD FUNCTIONAL COST SUMMARY TOTAL

	N_RECRNG	RECURRING	TOTAL
HOURS			
Engineering			
Research & Engineering	0	44,239	44,239
Electronic Systems	0	3,422	3,422
ElectronicSystems ICL	0	3,161	3,161
F-16 Program Mgmt.	0	9,741	9,741
F-16 Program Office	0	3,840	3,840
Logistics	659	29,941	30,600
Total Engr. Direct Hours	659	94,344	95,003
Manufacturing			
Tool Engineering	0	16,562	16,562
Tool Manufacturing Airframe	0	8,000	8,000
Tool Manufacturing Mse.	0	3,014	3,014
Plant Engineering	215	91	306
Factory Fabrication	0	39,116	39,116
Factory Assembly	0	86,155	86,155
Electronic Bench	0	33,232	33,232
Field Operations	0	24,370	24,370
Mod & Test Aircraft	0	307	307
Quality Assurance	0	725	725
Total Mfg. Direct Hours	215	211,572	211,787

REPORT GENERATED FROM COLUMN = PRRE

(Production/Retrofit)

08/27/90 15:07:49

General Dynamics

Fort Worth Divison – Fort Worth, Texas

STANDARD FUNCTIONAL COST SUMMARY TOTAL

	PRDCTION	RETROFIT	TOTAL
HOURS			
Engineering			
Research & Engineering	44,139	100	44,239
Electronic Systems	3,422	0	3,422
ElectronicSystems ICL	3,161	0	3,161
F-16 Program Mgmt.	9,741	0	9,741
F-16 Program Office	3,840	0	3,840
Logistics	30,600	0	30,600
Total Engr. Direct Hours	94,903	100	95,003
Manufacturing			
Tool Engineering	16,562	0	16,562
Tool Manufacturing Airframe	8,000	0	8,000
Tool Manufacturing Mse.	3,014	0	3,014
Plant Engineering	306	0	306
Factory Fabrication	39,116	0	39,116
Factory Assembly	86,155	0	86,155
Electronic Bench	33,232	0	33,232
Field Operations	24,370	0	24,370
Mod & Test Aircraft	307	0	307
Quality Assurance	725	0	725
Total Mfg. Direct Hours	211,787	0	211,787

REPORT GENERATED FROM COLUMN = YEAR (CALENDAR YEAR)

08/27/90 15:07:49

General Dynamics

Fort Worth Divison – Fort Worth, Texas

STANDARD FUNCTIONAL COST SUMMARY TOTAL

	__1988	__1989	__1990	__1991	__1992	TOTAL
HOURS						
Engineering						
Research & Engineering	0	337	4,669	23,819	15,414	44,239
Electronic Systems	0	0	0	379	3,043	3,422
ElectronicSystems ICL	0	0	0	1,283	1,878	3,161
F-16 Program Mgmt.	0	0	0	1,048	8,693	9,741
F-16 Program Office	0	0	0	413	3,427	3,840
Logistics	246	91	5,784	10,017	14,452	30,600
Total Engr. Direct Hours	246	428	10,463	36,959	46,907	95,003
Manufacturing						
Tool Engineering	0	0	212	14,319	2,031	16,562
Tool Manufacturing Airframe	0	0	102	6,910	988	8,000
Tool Manufacturing Mse.	0	0	354	2,128	532	3,014
Plant Engineering	0	0	34	204	68	306
Factory Fabrication	0	0	1,855	37,033	228	39,116
Factory Assembly	0	0	1,716	74,149	10,297	86,155
Electronic Bench	0	0	229	32,609	394	33,232
Field Operations	0	0	996	11,540	11,864	24,370
Mod & Test Aircraft	0	0	1	203	103	307
Quality Assurance	0	0	0	78	647	725
Total Mfg. Direct Hours	0	0	5,469	179,116	27,152	211,787

EXPLANATION OF MACRO TECHNIQUES USED IN PROGRAM

1. Assigning a Macro variable with a %LET statement

%LET COLUMNS = YEAR;

A character string can be directly assigned to a Macro variable using the %LET statement. %LET statements can appear anywhere in SAS program. See pages 27 and 87-88, *SAS Guide to Macro Processing, Version 6, Second Edition*.

2. Simple Macro variable references.

&COLUMNS

Whenever the Macro processor sees an ampersand (&) joined immediately to a character string, it searches the appropriate symbol table for a Macro variable of that name and substitutes the character string stored there for the Macro variable. Macro Variables can be used anywhere in a SAS program. See pages 29-30, *SAS Guide to Macro Processing, Version 6, Second Edition*.

3. Assigning a Macro variable with a CALL SYMPUT function

CALL SYMPUT(VAR, YEAR)

CALL SYMPUT('TOT_COLS', TOT_COLS)

CALL SYMPUT (argument1, argument2) assigns the string value in argument2 to the Macro variable in argument1. Argument1 may be a quoted string, such as 'TOT_COLS', or it may be a SAS variable containing a character string, such as VAR, which contains the strings 'VAR1', 'VAR2', etc. Argument2 may be a quoted character string or a SAS variable. CALL SYMPUT can only be used in a Data Step. See pages 165-172, *SAS Guide to Macro Processing, Version 6, Second Edition*.

4. Iterative %DO statements.

%DO J = 1 %TO &TOT_COLS;

&&VAR&J = .;

%END;

An iterative %DO statement repeats the actions inside the loop until the control variable of the loop equals the upper bound of the loop. It has a built-in counter that increments the value of the control loop after each pass through the loop. In its example where &TOT_COLS = 6, the &&VAR&J will be resolved six times and the strings resulting from '&&VAR&J = .;' will be emitted six times. See pages 66-68, *SAS Guide to Macro Processing, Version 6, Second Edition*.

5. %IF-%THEN conditional execution.

%IF &J NE 1 %THEN

%DO;

ELSE

%END;

SAS statements located within an %IF-%THEN statement are emitted only when that statement evaluates to true. In this example, the ELSE statement will be released only when &J is not equal to 1. See pages 15-16 and 77-78, *SAS Guide to Macro Processing, Version 6, Second Edition*.

6. %SUBSTR function.

%SUBSTR(&&VAR&J, 2, 4)

The %SUBSTR(argument, starting position, length) function returns a substring of characters. It begins its

substring operation at the starting position identified in the function and continues for a length specified in the function. If &&VAR&J resolves to __1990, the returned string would be 1990. See pages 128-129, *SAS Guide to Macro Processing, Version 6, Second Edition*.

7. %LENGTH function.

%LENGTH(&&VAR&J)

The %LENGTH(argument) function returns the number of characters in the resolved value of the argument it is examining. If the resolved value is __1990, the length would be five (5). See pages 120-121, *SAS Guide to Macro Processing, Version 6, Second Edition*.

8. %EVAL function

%EVAL(%LENGTH(&&VVAR&J)-1)

The %EVAL function performs simple integer arithmetic that does not involve integers greater than 2**32-1. In this example, it subtracts one (1) from the integer value returned from the %LENGTH function. Here, that would be 5, so the value returned by the %EVAL function would be 4 (5-1). See page 119, *SAS Guide to Macro Processing, Version 6, Second Edition*.

9. Double resolution of Macro variables.

%DO J = 1 %TO &TOT_COLS;

&&VAR&J = .;

%END;

Double resolution results from the Macro processor's orientation to change double ampersands to a single ampersand (&) during each scanning of the tokens and then to rescan any strings resulting from a resolution. Thus, a resolved string is scanned at least twice.

When J = 1, &&VAR&J resolves to &VAR1 after the first pass of tokenization. The && becomes a single &, the VAR remains as is, and the &J becomes a 1, creating &VAR1. Then, during the second pass, &VAR1 resolves to the string stored in the symbol table for that variable, which, in this example, is the SAS variable __1990. No further resolution occurs during the third pass, so the complete string emitted by this %DO loop would be '__1990 = .;'. %DO loops can occur only within the Macro code of an SAS program. See pages 32 and 36, *SAS Guide to Macro Processing, Version 6, Second Edition*.

10. %DO %WHILE loops.

%LET J = 1;

%DO %WHILE(&&SORT&J NE);

%LET LASTSORT = &&SORT&J;

%LET J = %EVAL(&J + 1);

%END;

A %DO %WHILE loop repeats the actions within the loop while a condition holds true. The evaluation occurs at the top of the loop, so the actions of the loop are never performed if the condition is untrue initially. Usually, %DO %WHILE loops are used with a counter that is explicitly initialized before the loop is started and explicitly incremented at the bottom of the loop before the control condition is evaluated. Like iterative %DO loops, %DO %WHILE loops can occur only within the Macro code of an SAS program. See page 70, *SAS Guide to Macro Processing, Version 6, Second Edition*.

SAS CODE GENERATED BY MACRO CODE

(See Section on Actions Performed in Program)

MACRO CODE

RESOLVED SAS CODE

1. The user's selection of the column variable of his report is stored in the Macro variable &COLUMN using a %LET statement. A blank entry will cause the program to fail. In this example, the SAS variable YEAR is loaded into &COLUMN.

```
%LET COLUMN = YEAR;
```

&COLUMN will resolve to YEAR.

The user's selection of the logical sort variables of his report are stored in the Macro variables &SORT1 - &SORT5. Blank entries are acceptable. They will resolve to blank during code generation. In this example, organization is loaded into the &SORT1 variable. Blanks are loaded into &SORT2 through &SORT5.

```
%LET SORT1 = ORG;
%LET SORT2 = ;
%LET SORT3 = ;
%LET SORT4 = ;
%LET SORT5 = ;
```

&SORT1 &SORT2 &SORT3 &SORT4 &SORT5 will resolve to
ORG (blank) (blank) (blank) (blank).

2. The report's data is sorted and summed by the page-sorting variables, the fixed row variable of the report (CEC) and the column variable selected by the user. That summed data is placed in the data set SUMDATA. The order of that data corresponds to the order needed in the report to be generated by the program. SPECDATA is also created in the same Data Step. It contains only the data values of the selected column variable and will be used to create the MACRO variables that will generate the new SAS column variables of the report.

```
/* SORT DATA IN PREPARATION OF SUMMING */
PROC SORT DATA = COSTDATA; /* INPUT DATA */
  BY &SORT1 &SORT2 &SORT3 &SORT4 &SORT5 /* PAGE SORTS */
  CEC /* FIXED ROW VARIABLE OF REPORT */
  &COLUMN; /* COLUMN VARIABLE OF REPORT */
RUN;

/* SUM DATA TO UNIQUE VALUES */
DATA SUMDATA(DROP=AMT) /* DATA TO BE REPORTED */
  SPECDATA(KEEP=&COLUMN); /* DATA TO GENERATE COL VARS */
  SET COSTDATA; /* SORTED INPUT DATA */
  BY &SORT1 &SORT2 &SORT3 &SORT4 &SORT5 /* PAGE SORTS */
  CEC /* FIXED ROW VARIABLE OF REPORT */
  &COLUMN; /* COLUMN VARIABLE OF REPORT */
  AMT + AMOUNT;
  IF LAST.&COLUMN THEN /* LAST OCCURENCE OF EACH DATA VALUE */
  DO;
    AMOUNT = AMT; /* MOVE SUMMED VALUE INTO AMOUNT */
    AMT = 0;
    OUTPUT; /* OUTPUT SUMMED VALUES */
  END;
RUN;
```

```
PROC SORT DATA = COSTDATA;
  BY ORG /* &SORT1 HAS RESOLVED TO ORG */
  CEC
  YEAR; /* &COLUMN HAS RESOLVED TO YEAR */
RUN;

DATA NEWDATA(DROP=AMT)
  SPECDATA(KEEP=YEAR); /* &COLUMN RESOLVED TO YEAR */
  SET COSTDATA;
  BY ORG /* &SORT1 RESOLVED TO ORG */
  CEC
  YEAR; /* &COLUMN RESOLVED TO YEAR */
  AMT + AMOUNT;
  IF LAST.YEAR THEN /* &COLUMN RESOLVED TO YEAR */
  DO;
    AMOUNT = AMT;
    AMT = 0;
    OUTPUT;
  END;
RUN;
```

3. SPECDATA, the data set containing only the data values of the selected column variable, is sorted by those data values. Then, in the subsequent Data Step, Macro variables of each of the data values of the column variable are created using CALL SYMPUT. Each Macro variable begins with the string 'VAR' and ends in an integer representing the sort order of the column attribute to be loaded into that variable. For example, if the column variable is YEAR and the data values of YEAR are 1990, 1991, 1992, 1993, 1994, and 1995, Macro variables VAR1 through VAR5 will be loaded with 1990 through 1995. An underscore (_) will be added to the beginning of each string, since SAS variable names cannot begin with integers. Also, using CALL SYMPUT, the total number of columns that will be printed will be loaded at the end-of-file to the Macro variable &TOT__COLS. The value of &TOT__COLS will be used to control the interactive %DO loops that will generate the SAS code that sums all column variables into the TOTAL variable and that writes the column data and headings during the DATA __NULL__ print step;

```
/* SORT DATA VALUES OF COLUMN VARIABLE */
PROC SORT DATA = SPECDATA;
  BY &COLUMN;
RUN;

/* CREATE MACRO VARIABLES FOR COL VARIABLES OF REPORT */
DATA __NULL__;
  SET SPECDATA END=EOF; /* DATA TO GENERATE COL VARIABLES */
  BY &COLUMN;
  IF FIRST.&COLUMN THEN /* FIRST OCCURENCE OF EACH VALUE */
  DO;
```

```
PROC SORT DATA = SPECDATA;
  BY YEAR; /* &COLUMN HAS RESOLVED TO YEAR */
RUN;

DATA __NULL__;
  SET SPECDATA END=EOF;
  BY YEAR; /* &COLUMN HAS RESOLVED TO YEAR */
  IF FIRST.YEAR THEN
  DO;
```

MACRO CODE

RESOLVED SAS CODE

```

/* INCREMENT J BY ONE FOR EACH UNIQUE DATA VALUE */
J + 1;
/* CONVERT J TO AN ALPHANUMERIC. LOAD TO TOT__COLS. */
TOT__COLS = LEFT(PUT(J,2));
/* BUILD NAMES OF MACRO VARIABLES BY CONCATENATING */
/* TOT__COLS TO THE END OF THE STRING 'VAR' AND THEN */
/* LOADING THAT RESULTING STRING TO THE VARIABLE VAR. */
VAR = 'VAR' || TOT__COLS;
/* LOAD THE NEW SAS COL VARIABLE NAMES TO MACRO */
/* VARIABLES. PREFIX NUMERIC VALUES WITH AN '_' */
/* IN ORDER THAT THE VARIABLE NAME WILL BE LEGAL. */
/* THE RESULTING MACRO VARIABLES WILL BE &VAR1, &VAR2, */
/* &VAR3, ETC. */
IF SUBSTR(&COLUMN,1,1) GE '0' AND
SUBSTR(&COLUMN,1,1) LE '9' THEN
    CALL SYMPUT(VAR,('_' || &COLUMN));
ELSE
    CALL SYMPUT(VAR,&COLUMN);
END;
/* LOAD TOTAL NUMBER OF COLUMN VARIABLES TO &TOT__COLS */
/* FOR USE IN CONTROLLING ITERATIVE %DO LOOPS. */
IF EOF THEN
    CALL SYMPUT('TOT__COLS',TOT__COLS);
RUN;

```

```

J + 1;

TOT__COLS = LEFT(PUT(J,2));

VAR = 'VAR' || TOT__COLS;

/* &COLUMN HAS RESOLVED TO YEAR */
IF SUBSTR(YEAR,1,1) GE '0' AND
SUBSTR(YEAR,1,1) LE '9' THEN
    CALL SYMPUT(VAR,('_' || YEAR));
ELSE
    CALL SYMPUT(VAR,YEAR);
END;

IF EOF THEN
    CALL SYMPUT('TOT__COLS',TOT__COLS);
RUN;

```

4. The next Data Step transposes the data, creating numeric SAS column variables from the data values of the selected column variable. An iterative %DO loop generates IF-THEN-ELSE statements that assign the data values of the column variable to new variables having the same name as the data value. Beginning underscores (__) are excluded from the evaluation since they were added to the beginning of the Macro variable in order that the forthcoming SAS variable would have a legal SAS name.

```

/* TRANSPOSE DATA FOR REPORT */
DATA RPTDATA(DROP=AMOUNT &COLUMN); /* TRANSPOSED DATA SET */
SET SUMDATA END=EOF; /* SORTED AND SUMMED DATA */
BY &SORT1 &SORT2 &SORT3 &SORT4 &SORT5 CEC;
/* CREATE A COLUMN VARIABLE FOR EACH DATA VALUE */
/* BY ASSIGNING THE NUMERIC TOTAL FOR THAT DATA VALUE TO */
/* THE RELATED COL VARIABLE USING IF STATEMENTS JOINED */
/* TO ASSIGNMENT STATEMENTS. */
%DO J = 1 %TO &TOT__COL;
/* GENERATE AN ELSE STATEMENT FOR EACH IF STATEMENT */
/* EXCEPT THE FIRST. */
%IF &J NE 1 %THEN
    %DO;
        ELSE
    %END;
/* DROP BEGINNING UNDERSCORES (__) FOR THE */
/* EVALUATION USED TO CREATE NEW COLUMN VARIABLES.*/
%IF %SUBSTR(&&VAR&J,1,1) = _ %THEN
    %DO;
        %LET DISTANCE = %EVAL(%LENGTH(&&VAR&J)-1);
        %LET READVAR = %SUBSTR(&&VAR&J,2,&DISTANCE);
    %END;
%ELSE
    %LET READVAR = &&VAR&J;
/* GENERATE THE IF-STATEMENTS TO ASSIGN THE */
/* NUMERIC VALUES TO THE NEW COLUMN VARIABLES. */
IF &COLUMN = "&READVAR" THEN
    &&VAR&J = AMOUNT;
%END;
/* AT THE LAST OCCURRENCE OF EACH DATA VALUE OF THE */
/* FIXED ROW VARIABLE, OUTPUT ALL COL VARIABLES TO THE */
/* TRANSPOSED DATA SET. THEN SET THEIR VALUES TO MISSING. */

```

```

DATA RPTDATA(DROP=AMOUNT YEAR); /* &COL RESOLVED TO YEAR */
SET SUMDATA END=EOF;
BY ORG CEC; /* &SORT1 RESOLVED TO ORG */

/* IFTHEN-ELSE STATEMENTS TO CREATE SAS VARIABLES */
/* FROM THE UNIQUE DATA VALUES OF THE SELECTED COLUMN */
/* VARIABLE HAVE BEEN GENERATED. */
IF YEAR = "1990" THEN
    __1990 = AMOUNT;
ELSE IF YEAR = "1991" THEN
    __1991 = AMOUNT;
ELSE IF YEAR = "1992" THEN
    __1992 = AMOUNT;
ELSE IF YEAR = "1993" THEN
    __1993 = AMOUNT;
ELSE IF YEAR = "1994" THEN
    __1994 = AMOUNT;
ELSE IF YEAR = "1995" THEN
    __1995 = AMOUNT;

```

MACRO CODE

```
IF LAST.CEC THEN
DO;
  OUTPUT;
  /* GENERATE SAS CODE TO SET VARIABLES TO MISSING. */
  %DO J = 1 %TO &TOT__COL;
    &&VAR&J = .;
  %END;
```

```
END;
RUN;
```

- 5A. In the Data __Null__ Step that writes the report, a new logical page is printed each time there is a change in the data value of the last sort variable specified. The Macro code uses a %DO %WHILE loop to determine the last sort variable specified by the user.

```
DATA __NULL__; /* REPORT-WRITING STEP */
  SET RPTDATA; /* TRANSPOSED REPORT DATA */
  FILE PRINT;
  BY &SORT1 &SORT2 &SORT3 &SORT4 &SORT5 CEC;
  %LET LASTSORT = ; /* SET LASTSORT TO BLANK */
  %LET J = 1; /* INITIALIZE J TO ONE (1) */
  /* DO WHILE SORT VARIABLES ARE NOT BLANK */
  %DO %WHILE (&&SORT&J NE );
    /* LOAD SORT VARIABLE INTO LASTSORT */
    %LET LASTSORT = &&SORT&J;
    %LET J = %EVAL(&J + 1); /* ADD ONE (1) TO J */
  %END;
  /* IF &LASTSORT IS NOT EQUAL TO BLANK, THEN ISSUE */
  /* THE SAS CODE TO PRINT A NEW LOGICAL PRINT PAGE. */
  %IF &LASTSORT NE %THEN
    %DO;
      IF FIRST.&LASTSORT THEN
        PUT __PAGE__;
    %END;
```

- 5B. In the Data __Null__ Step, a TOTAL variable is created by a SUM of the column variables generated by an iterative %DO loop. That loop is controlled by the &TOT__COL value that represents the number of column variables which were created in the TRANSPOSE step. The variables are generated using double resolution.

```
TOTAL = SUM(OF
  /* GENERATE SAS VARIABLES TO BE SUMMED INTO TOTAL */
  %DO J = 1 %TO &TOT__COL;
    &&VAR&J
  %END;
  );
```

- 5C. In the Data __Null__ Step, the left-hand descriptive variable and the generated SAS column variables are written using an iterative %DO loop controlled by &TOT__COL. The pointer position for the PUT statements is controlled by &POSITION, which is initialized to 35 and advanced by 12 (the length of the numeric fields + 1) for each pass through the loop. The %EVAL function is used to add 12 to &POSITION.

```
%LET POSITION = 35;
PUT @ 3 CEC $CEC. /* LEFT-HAND DESCRIPTIVE VARIABLE */
%DO J = 1 %TO &TOT__COL;
  /* PRINT LOCATION AND FORMAT OF EACH COLUMN */
  @&POSITION &&VAR&J COMMA11.
  /* CALCULATE NEXT PRINT POSITION */
  %LET POSITION = %EVAL(&POSITION + 12);
%END;
  /* PRINT POSITION AND FORMAT OF TOTAL COLUMN */
  @&POSITION TOTAL COMMA13.;
```

RESOLVED SAS CODE

```
IF LAST.CEC THEN
DO;
  OUTPUT;
  /* SAS CODE HAS BEEN GENERATED */
  /* TO SET THE NEW COLUMN VARIABLES TO */
  /* MISSING (.) AFTER OUTPUT. */
  __1990 = .;
  __1991 = .;
  __1992 = .;
  __1993 = .;
  __1994 = .;
  __1995 = .;
```

```
END;
RUN;
```

```
DATA __NULL__;
  SET RPTDATA;
  FILE PRINT;
  BY ORG CEC; /* &SORT1 RESOLVED TO ORG */
```

```
/* &LASTSORT RESOLVED TO ORG */
IF FIRST.ORG THEN
  PUT __PAGE__;
```

```
TOTAL = SUM(OF __1991 __1991 __1992 __1993 __1994 __1995);
```

```
PUT @ 3 CEC $CEC30.
  @ 35 __1990 COMMA11. @ 47 __1991 COMMA11.
  @ 59 __1992 COMMA11. @ 71 __1993 COMMA11.
  @ 83 __1994 COMMA11. @ 95 __1995 COMMA11.
  @ 107 TOTAL COMMA13.;
```

- 5D. In the HEADER area of the Data __Null__ Step, the character string 'TOTAL' is printed just below the titles of the report if &LASTVAR is blank, which occurs when the user does not specify one or more sort variables. Otherwise, the values of the page-sort variables are printed just below the report's titles.

```
%IF &LASTVAR = %THEN
  %DO;
    PUT      @18 'TOTAL';
  %END;
%ELSE
  %DO;
    PUT      @18 &SORT1  @50 &SORT2 /
             @18 &SORT3  @50 &SORT4 /
             @18 &SORT5  ;
  %END;
/* ORG WAS SPECIFIED AS &SORT1 */
PUT @18 ORG //;
```

- 5E. In the HEADER area of the Data __Null__ Step, the descriptive headings of each numeric column are generated using an iterative %DO loop controlled by &TOT__COLS. &POSITION is again used to control the pointer position for the PUT statement that writes the descriptive headings. Here, however, it is used with the %LENGTH value of the character string that is to be printed. The length of character string is subtracted from 11 (the numeric field width) and that difference is added to the &POSITION value using the %EVAL function. For example, if the string were '__1990', the %LENGTH value would be 5 and, thus, 6 (11-5) would be added to the value of &POSITION. Adding 6 to &POSITION places the '__1990' string at the right-most 5-character position of the numeric field.

```
%LET POSITION = 35;
PUT
/*DO J = 1 %TO &TOT__COL;
/* CALCULATE LENGTH OF HEADING STRING */
%LET Y = %LENGTH(&&VAR&J);
/* CALCULATE STARTING LOCATION OF HEADING STRING */
%LET START = %EVAL(&POSITION+(11-&Y));
/* ISSUE HEADING STRING TO BE WRITTEN TO PRINT FILE */
@&START "&VAR&J"
/* DETERMINE LOCATION OF NEXT HEADING STRING */
%LET POSITION = %EVAL(&POSITION + 12);
%END;
/* LOAD LENGTH OF TOTAL STRING TO Y */
%LET Y = 5;
/* CALCULATE STARTING LOCATION OF TOTAL STRING */
%LET Z = %EVAL(&POSITION+(13-&Y));
/* ISSUE TOTAL STRING */
@&Z "TOTAL";

PUT @35 "__1990" @ 53 "__1991"
   @ 66 "__1992" @ 78 "__1993"
   @ 90 "__1994" @102 "__1995"
   @114 "TOTAL";
```