What are coding conventions, and why use them? Coding conventions are a set of standards used in writing code adopted by a programmer or a group of programmers. The purpose of these coding conventions is to help make software more reliable, maintainable, efficient and self-documenting. The SAS program below and the three macros it calls, which are also displayed, illustrate the use of coding conventions. If this were a production application, the macros would be defined at the beginning of the program, and the open SAS code would follow. Instead, for demonstration purposes, the macro definitions follow the open SAS code.

First, a simple SAS dataset is created in open code. Then the CREAFMT macro is called twice: once to build a numeric format and a second time, a character format. Each format is stored in a separate SAS library specified as a parameter to the macro call. The macro CONCFMT then copies both formats into the WORK library to make them available at the same time. A DATA step uses these formats to build a second data set. Finally, the TESTPRINT macro is called to demonstrate that the formats just created are available to the calling program.
This SAS code illustrates a number of coding standards. Among them:

* Put a comment block similar to the one above at the beginning of each module. It is advisable for each project, group and/or installation to adopt a standard comment block style. Other items which may appear in this block are lists of inputs and outputs, the date(s) updated, a summary of changes made and a list of macros or other modules invoked.

* Put no more than one statement per line.

* Put a RUN statement, followed by a blank line immediately after every DATA and PROC step. The RUN statement defines a step boundary explicitly and the blank line adds readability and clarity. Blank lines should also be used within steps to set off blocks of code.

Use a consistent indentation scheme.

Group related statements together, such as the FILENAME and LIBNAME statements above, to make the code easier to read and maintain.

Line up variable names and formats, as on the INPUT statement above. Putting one variable per line simplifies code maintenance. If the semicolon which ends the INPUT statement is on a separate line, the statement will be easier to modify in the future.

Use meaningful names for variables, datasets and libraries.
Note that when the TESTPRNT macro is invoked, PROC PRINT will only be executed if the global macro variable DEBUG has the value YES. Thus the calls to the TESTPRNT macro can remain in the software through its life-cycle, ready to serve a built-in debugging function in later maintenance and enhancement.

@MACRO CREAFMT (DATA=_LAST_, / * input data set */ START=, / * dataset variable to format */ LABEL=, / * dataset variable with label */ FMTNAME=, / * name of output format */ FMTLIB=WORK); / * library for output format */

@PUT %STR( ); /* a blank line on log for readability */
@PUT *** NOTE: MACRO CREAFMT IS BEGINNING EXECUTION.;
@PUT **( ) DATA=&DATA START=&START LABEL=&LABEL
FMTNAME=&FMTNAME FMTLIB=&FMTLIB;
@PUT %STR( );

**************************************************************************
* NAME: CREAFMT *
* STORED: C:\SOGI\CREAFMT.SAS *
* PROD DATE: 08/23/91 *
* AUTHORSHIP: Atlantic Research Corporation *
* FUNCTION: This utility macro creates a format from a *
* dataset where one variable ("START") holds *
* values to be formatted and another variable *
* ("LABEL"), values of the corresponding label. *
* If "START" is a character variable, the name *
* of the format created must start with a *
* dollar sign ($).
* ASSUMPTIONS: If a non-default value of the "FMTLIB=" *
* parameter is passed to this macro, it has *
* been defined previously.
* The dataset name _TEMP used in this macro *
* is not used elsewhere in the calling program.*
* REFERENCE: I. CANCODE, (301) 258-5300 *
**************************************************************************;

DATA _TEMP (RENAMEx(&START=START &LABEL=LABEL));
SET &DATA (KEEP=&START &LABEL);
RETAIN FMTNAME = &FMTNAME;
RUN;
PROC FORMAT CNTLIN= __
LIBRARY=&FMTLIB;
RUN;

@PUT %STR( ); /* a blank line on log for readability */
@PUT *** NOTE: MACRO CREAFMT HAS COMPLETED EXECUTION.;
@PUT %STR( );
@MEND CREAFMT;

@PUT %STR( ); /* a blank line on log for readability */
@PUT *** NOTE: MACRO CREAFMT IS NOW COMPILED.;
@PUT %STR( );

In the CREAFMT and subsequent macro definitions, the comment block is similar to the one used in the calling program, except that macro comments (%* ... ;) are used, rather than the slash and asterisk (/* ... */) comments used in open code. Other items such as INPUTS, OUTPUTS, CALLED BY, and CALLS TO could also be used in the comment block, depending on the application and the function of the macro. Here, CALLED BY or CALLS TO are unnecessary as this is a commonly-called utility function.

The use of meaningful variable names makes this macro as self-documenting as possible. Coding the name of the macro on the %MEND statement serves as documentation and promotes clarity.

Note that each parameter in the %MACRO statement is on a separate line with a slash/asterisk style comment to describe its use and, possibly, expected values. To be as consistent as possible with the SAS System defaults, DATA= is used as the parameter name of the input dataset. DATA=_LAST_ is the SAS system default in a PROC or SET statement if a dataset name is not specified. The CNTLIN option of PROC FORMAT expects to find SAS variables named START and LABEL.

The %PUT ***NOTE: and %PUT %STR( ) statements are built-in debugging features to make the SAS log more readable. The NOTES/NOOTES SAS option also controls the display of %PUT text which contains the string 'NOTES'. The three asterisks before NOTE: distinguish it from standard SAS notes. The blank line between macros provided by %PUT %STR puts white space between macros on the SAS log. A %PUT statement displaying
the values of the keyword parameters passed into the macro is another useful built-in debugging tool. Though the %PRINT, %SYMBOLGEN and %MLOGIC options provide similar functions and should be used, explicit %PUT statements can customize SAS Log messages.

%MACRO CONCFMT
(INLIBS = , /* List of librefes of input libraries */
OUTLIB = WORK; /* Libref of output library */

%PUT %STR( );
%PUT ***NOTE: MACRO CONCFMT IS BEGINNING EXECUTION;
%PUT %STR( ) INLIBS = &INLIBS OUTLIB = &OUTLIB;
%PUT %STR( );

%LOCAL I PIECE;
PROC CATALOG CAT = WORK.FORMATS; /* default libref and catalog */
/* Parse &INLIBS and generate one run group for each token */
%LET I = 1;
%DO %UNTIL (%SCAN (&INLIBS, &I, %STR( ) = ) = ); /* for each libref */
%LET PIECE = %SCAN(&INLIBS, &I, %STR( ));
/*
COPY OUT = &OUTLIB.FORMATS /* default libref/catalog */
   IN = &PIECE.FORMATS /* libref/catalog input */
RUN; /* end of run group, not PROC CATALOG */
%LET I = %EVAL(&I + 1); /* find next libref in list */
%END;
QUIT; /* ends proc catalog */
%PUT %STR( );
%PUT ***NOTE: MACRO CONCFMT HAS COMPLETED EXECUTION;
%PUT %STR( );
%MEND CONCFMT;

%PUT %STR( );
%PUT ***NOTE: MACRO CONCFMT IS NOW COMPILED;
%PUT %STR( );

It is good practice to declare all macro variables local or global as above. Identical comments on the first and last statements of a performance group (e.g. %DO...%END, %DO...%END).

Aligning these statements is advisable, especially when there are many levels of nesting. Note that in this macro there is macro code within a PROC step and that SAS code and macro code follow independent indentation schemes.
%MACRO TESTPRNT (DATA=LAST, VARLIST=ALL, IDLIST=-, NLINE=2, OBS=500, REPORT=NO); /* input data set */ /* variables to print */ /* ID variables, if any */ /* number of last title line in */ /* the calling environment which */ /* should not be overridden */ /* number of observations to print */ /* pass in MAX to print them all */ /* if YES, print regardless of */ /* DEBUG value */
%PUT %STR( );
%PUT NOTE: *** MACRO TESTPRNT IS BEGINNING EXECUTION;
%PUT %STR( ) DATA = &DATA VARLIST = &VARLIST;
%PUT %STR( ) IDLIST = &IDLIST NLINE = &NLINE;
%PUT %STR( ) OBS = &OBS REPORT = &REPORT;
%PUT %STR( );

******************************************************************************
** NAME: TESTPRNT
** STORED: C:\SUGI\TESTPRNT.SAS
** PROD DATE: 09/01/91
** AUTHORSHIP: Atlantic Research Corporation
** FUNCTION: This macro performs a PROC PRINT of a SAS dataset if the macro variable DEBUG in the calling environment (not a parameter) has the value "Y", "YES" or "DEBUG" or if the parameter REPORT= has the value "Y" or "YES". If "DATA=" is not set in the environment, this macro uses the most recently created dataset. If the parameter OBS= is not passed in the macro call, the first 500 observations are printed. Two title lines are printed. The first line, TITLE(&TITLN+1), shows the maximum number of observations printed ("ALL" if &OBS=MAX); the second title line shows the dataset name. ASSUMPTION: There is a global macro variable &DEBUG in the calling environment.
** UPDATED: 08/19/90
** REFERENCE: A. SASWHIZ (301) 258-2666
******************************************************************************

%LOCAL__DSN /* specified dataset name or last dataset created */
__OBSOPT /* number of observations specified, or a blank */

/* determine whether to perform PROC PRINT */
%IF (%UPCASE(&DEBUG)=DEBUG OR %UPCASE(&DEBUG)=Y OR %UPCASE(&DEBUG)=YES)
OR (%UPCASE(&REPORT)=YES OR %UPCASE(&REPORT)=Y) %THEN

%DO; /* execute test print */;

%IF &DATA = LAST %THEN /* determine name of last dataset created */
%LET __DSN = %SCAN(&SYSDSN,1)%STR(.)%SCAN(&SYSDSN,2);

%ELSE %LET __DSN=&DATA;

/* prepare an OBS= option for the dataset, if necessary */
%IF %UPCASE(&OBS) = MAX %THEN
%LET __OBSOPT = ;
%ELSE %LET __OBSOPT = (OBS=&OBS);

/* print the data */
PROC PRINT DATA=&__DSN & __OBSOPT SPLIT=';';
VAR &VARLIST;
ID &IDLIST;
/* use macro code to build the TITLE statements */
%IF %UPCASE(&OBS) = MAX %THEN
%LET NUMOBS = ALL;
%ELSE %LET NUMOBS = UP TO &OBS;
The use of keyword, rather than positional, macro parameters is strongly recommended. The TESTPRNT macro shows that using keyword parameters simplifies macro calls and allows more flexibility in calling a macro with many parameters. Common-sense default values were assigned to the macro parameters so that the user needs to specify a parameter only to override the default value. Had the six parameters been positional, omission of any one parameter could easily cause errors and produce unintended results.

In conclusion, the examples in this paper represent a mere sampling of possible conventions to help programmers produce clear, easily maintainable code with built-in error-prevention and debugging capacity.

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and in other countries. © indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

The authors can be contacted at:

Atlantic Research Corporation, Information Systems Division
1301 Piccard Drive, 2nd Floor
Rockville, MD 20850
(301) 258-5300

992