ABSTRACT
SAS® Macros and the SAS® Macro Language are simple yet powerful tools. Once you understand how to use and write macros, your next task is to understand how to use them consistently and more importantly, intelligently, from a programmer's point of view. This tutorial presents the basics of project design with an overview of macro usage across programs. Topics include communicating between programs, using system variables, use of the config.sas, autoexec.sas, %includes, etc. Expected audience are beginner and intermediate SAS programmers.

INTRODUCTION
This paper explains how I began using macros and have progressed over the years. I think it is important to illustrate the thought progression of beginning macro usage and progressing to more complicated and then back to more simple usage. The program examples illustrate the differing styles of using macros. Most have been generalized for this paper in order to highlight the concepts and the differences between user-written and data-supplied macro usage.

TABLE OF CONTENTS
* Eight D's of any Scientific Investigation
* programming considerations
  * naming conventions
  * directory structure
  * data dictionary and formats
* notes on a style sheet
* three steps in writing macros
  * write twice
  * parameterize
  * make macro
* getting started: %%global macro variable usage
  * autoexec
  * utilities: titles, nobs
* example using proc FREQ

8 D's OF SCIENTIFIC INVESTIGATION
Documentation:
Determine Goals
Study Design: data collection instrument
Data Collection: data dictionary
Data Entry
Data Management: SAS programs
Data Analysis: number-crunching, bean counting,
Disseminate Results: publish

Where do you work? Hopefully you're consulted on the design of the data collection instrument. Certainly you want to be familiar with the data dictionary, as it contains what you need to know to program: format definitions, variable type, length, and label.

PROGRAMMING CONSIDERATIONS
These programs are examples from a typical project. The main reason I learned to use macro variables was to be able to reuse programs developed in one project in the others. This exercise, in turn, lead me to consider the organizational programming issues of naming conventions and directory structure.

NAMING CONVENTIONS
I manage projects for two Research Groups:
* PEP: Performance Evaluation Project
* TB : Tuberculosis

Each of those groups has several projects
PEP: HIV: Human Immunodeficiency Virus
HTL: Human T-Lymphocyte Virus
TB: NAA: Nucleic Acid Amplification
NTM: Non-Tuberculosis Mycobacteria

These projects have biannual shipments, e.g.: NAA-2000-Jan NAA-2000-Aug

Over the years I developed these data set naming conventions:

<table>
<thead>
<tr>
<th>col</th>
<th>name</th>
<th>range</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ID:</td>
<td>P:</td>
<td>Panel</td>
</tr>
<tr>
<td>2</td>
<td>S:</td>
<td>Sample</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>yymm</td>
<td>Year+Month: 2000-Jan</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>data</td>
<td>A:</td>
<td>Study: A=Any-Other, E=EIA, ...</td>
</tr>
<tr>
<td>5</td>
<td>DT</td>
<td>D,T</td>
<td>data, text</td>
</tr>
<tr>
<td>6</td>
<td>ver</td>
<td>0:3</td>
<td>version: 0=final, 1=master, 2=clean, 3=with corrections</td>
</tr>
</tbody>
</table>

example:
P00001AD0 Panel data for Any-Other
P00001AD1\ P00001AD2 > versions one thru three
P00001AD3 /
P00001AT0 text data set
S00001AD0 Samples, transposed from Panel

See the macro TITLES below and the global macro variable DATA_SET which implements this data set naming convention.

DIRECTORY STRUCTURE
A directory for an individual shipment includes these sub-directories under MS-Windows:
c:\...\NAA\p0008 parent directory
..\doc data collection form, data dictionary
..\htm HyperText Mark-Up Language from SAS ODS
..\pgm SAS programs
..\ssd SAS data sets
..\xls Excel, written from summary data sets

Note that each directory contains a specific set of file types. See the use of the macro-variable PATH below in pointing to each directory.

DATA DICTIONARY
Early in my career I depended on a print-out of proc CONTENTS as my data dictionary. Developing a good workable data dictionary is very important. Getting the people whose data I manage to understand how important it is to them and that it should therefore be, not only their priority but also their responsibility occupied me in many meetings over several years. There are two very important items in a good data dictionary: format definitions and data definitions.
Here is an abbreviated example from one of my projects:

Data Dictionary: PEP HIV-1 Results Panel 2001-01
CDC: programmer/analyst: Ronald Fehd
Shipment ID : 9
Shipment date: January 29, 2001

Contents:
I. Sample Identification look-up table
II. Deliverables Description
III. Editing Instructions
IV. Formats
V. Variable Descriptions
...
IV. Formats

value HIV9SN /*Sample-Nmbr to Sample-Name */
0 = "&BLANK."
1 = '9-1'
...[snip]
5 = '9-5'
other = "&INVALID."

V. Variable Descriptions


MACRO VARIABLE USAGE
There are two ways of creating and initializing macro variables:
the first is a user-defined macro variable and the second is using
data to supply a value.

The simple and direct way to create a macro variable is:
%LET MACVAR = blank for now;

Note that the keyword and name are in upper case; this is a
reminder that this statement operates in the global programming
environment, the same as title, footnote, and option statements.

In addition, it is important to realize that every macro variable is a
character string, even when the value contains only digits and may
conform to the appearance of a real number:
%LET PI = 3.1416; %LET SUM = &PI. + 1;%PUT SUM<&SUM>;
yields this note in the log: SUM<3.1416 + 1>;
%LET SUM =%eval( &PI. + 1);%PUT SUM<&SUM>;
we're getting closer: SUM<4>; because the %eval function does
only integer arithmetic!

Remember: macro variable values are always strings!

The indirect way to create a macro variable from data is to use a pair of
statements:
1. call symput('MACVAR',<character expression>);
2. RUN;

The RUN statement is a step boundary, and only after a step
boundary is the macro variable available to the next step. See usage
in program INOBS, lines 8 and 9, below.

THREE STEPS IN WRITING MACROS
1. Recognize a pattern: In developing macros I notice when I have
written two similar SAS paragraphs: two similar sets of statements.
Once I have two different paragraphs working correctly, then I begin
to review and identify the commonalities and differences, which are
the parameters of the first version of a potentially reusable program.

2. Prepare a parameterized %include file. I use global macro
variables for the parameters. The major benefit of this intermediate
step, especially for beginners, is that you still have line numbers
available from your SAS log. Those line numbers are so valuable in
debugging because they tell you where your SAS errors are located.
This point is important! Line numbers are no longer available from
within macros!

3. Define a macro and convert the list of global macro variables used
to macro parameters. Kiss the line numbers in the log goodby; now
you have to guess where the errors are occurring: somewhere inside
your macro! These steps are illustrated in the examples below.
Project Set-up: three examples of autoexec.sas

```
01 /* AUTOEXEC -------------------------------------- */
02 libname LIBRARY 'C:\SAS\PEP\HIV\P200101\SSD';
03 TITLE1 HIV-1 shipment A: 2001-Jan-31;

This is as simple as it gets: where is the data stored and what is the main title to be used in reports.

```
01 /* AUTOEXEC basic -------------------------------- */
02 %LET PROJECT = HIV; %*(HIV,HTL,NAA,NTM): directory PATH-name ;
03 %LET PROJNAME = HIV-1; %*PROJECT expanded: TITLES ;
04 %LET SHIP_ID = A; %*(0:9,A:Z) : infix in format names;  
05 %LET SHIPYYMM = 200101; %*ccYYMM : names: data set, PATH;  
06 %LET SHIPDATE = Jan 29, 2001; %*Mon dd, ccYY : report TITLES ;
07 /* AUTOEXEC begin execution----------------------- */
08 TITLE "&PROJNAME. Shipment: &SHIP_ID: &SHIPYYMM. &SHIPDATE.";
09 %LET LIBRARY = LIBRARY; %*default library;  
10 %LET DATA_SET = A&SHIPYYMM.D0;%*default data set, reset by macro TITLES;  
11 %LET BLANK = BLANK; %*numeric: missing and character: blank;  
12 %LET INVALID = INVALID; %*out of expected range: see formats;  
13 %LET PATH = C:\SAS\PEP\&PROJECT.P&SHIPYYMM.;  
14 %LET PATH_HTM = &PATH.HTM; %*HyperText Markup;  
15 %PUT _USER_; %*checking;  
1616
17 libname LIBRARY "&PATH.SSD";
18 %INCLUDE SASAUTOS(Titles); %*load fan-in macro(s);  
19 options details noCenter source2;  
2020
21 run;%*.......................... AUTOEXEC end;  
```

Now we're using macro variables, perhaps a little too much!

This is a basic autoexec with global macro variables. It is useful when programming with parameterized %include files.

Refer to the data dictionary and note the project description macro variables defined in lines 02:06 are used in the title in line 08. Lines 11:12 are first used in a proc FORMAT program, – which would be cut and pasted from the data dictionary example above – later in data review, and finally in excluding observations before creating summary data sets. PATH is defined in line 13 and concatenated in references in line 14 and 17. Any other directory could be pointed to using this style; see reference to PATH_HTM in the FREQ1VAR ODS example and PATH_XLS in the macro SAS2XLS. The statement %PUT USER in line 15 shows the user-defined macro variables in the session. The option source2 in line 19 shows all lines from %included files, and is an essential aid in debugging.

```
01 /* AUTOEXEC intermediate -------------------------------- */
02 %LET PROJECT = HIV; %*(HIV,HTL,TB) : directory PATH-name;  
03 %LET PROJNAME = HIV-1; %*PROJECT expanded: TITLES;  
04 %LET SHIP_ID = A; %*(0:9,A:Z) : infix in format names;  
05 %LET SHIPYYMM = 200101; %*ccYYMM : names: data set, PATH;  
06 %LET SHIPDATE = Jan. 24, 2001;%*Mon dd, ccYY : report TITLES;  
07 /* AUTOEXEC begin execution----------------------- */
08 TITLE "&PROJNAME.: Shipment: &SHIP_ID: &SHIPYYMM. &SHIPDATE.";
09 09
10 %LET PATH = C:\SAS\PEP\&PROJECT.P&SHIPYYMM.;  
11 %LET LIBRARY = LIBRARY; %*default libname;  
12 %LET DATA_SET = A&SHIPYYMM.D0;%*default data set, reset by macro TITLES;  
13 %PUT _USER_; %*checking;  
1414
15 filename SASAUTOS("getoption(SASUSER)"");%*add program directory to search-path for macros;  
16 options details noCenter source2;  
1717
18 %TITLES(); %*checking;  
19 19
20 libname LIBRARY "&PATH.SSD";
2121
22 options details noCenter source2;  
23 dm log 'awsmaximize on;log;zoom on;up max;down max' log;%*view log;  
24 24
25 run;%*.......................... AUTOEXEC end;  
```

In this intermediate autoexec global macro variables have been moved to small files. See LetFrmt and LetPath below.

General purpose macros are stored in a separate file with the name of the macro as the file name, e.g.: the macro TITLES is stored in file TITLES.SAS. In line 15 the fileref SASAUTOS points to the program directory: SASUSER, which is defined in your SAS invocation. A review of your proc OPTIONS will show

sasautos=sasautos. The fileref SASAUTOS is the autocall library, that is it contains macros within same-named files; these macros are called autocall macros. The option MautoSource enables searching of the SASAUTOS fileref for the autocall macros which are then %included. See the macros Nobs and Freq1Var below.

In this intermediate autoexec global macro variables have been moved to small files. See LetFrmt and LetPath below.
Lest we forget: having identified BLANK, INVALID, and PATH_HTM as global macro variables used in few programs, remove them to their own files and include them only when needed.

The TITLES macro is a fan-in routine: it is called by many other programs. It serves two main purposes:

1. returns %global macro variable DATA_SET, declared in line 07 and set in line 10; it is concatenated from its various parts according to either your site's or the individual project's data set naming conventions, discussed above.

2. sets global environment variable title2. Note carefully that the statement begins on line 12 and ends with the second semicolon on line 20 in column 72. Note also that the macro control statements %IF and %ELSE in lines 13:19 do not generate any SAS semicolons; their only result is string literals for the title statement. The secondary effects – line 22 – are footnotes are cleared and page number is reset to one.

Further titles in a program may be placed after calling the TITLES macro using this example statement:

<table>
<thead>
<tr>
<th>1</th>
<th>/<em>letFmt %GLOBAL mac-vars used in formats, exception reports, summary</em>/</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>%LET BLANK = BLANK; %LET INVALID = INVALID;</td>
</tr>
<tr>
<td>1</td>
<td>/*letPath %GLOBAL mac-vars used in writing reports to other files */</td>
</tr>
<tr>
<td>2</td>
<td>%LET PATH_HTM = &amp;PATH.HTM; %LET PATH_XLS = &amp;PATH.XLS;</td>
</tr>
</tbody>
</table>

The TITLES macro is a fan-in routine: it is called by many other programs. It serves two main purposes:

1. returns %global macro variable DATA_SET, declared in line 07 and set in line 10; it is concatenated from its various parts according to either your site's or the individual project's data set naming conventions, discussed above.

2. sets global environment variable title2. Note carefully that the statement begins on line 12 and ends with the second semicolon on line 20 in column 72. Note also that the macro control statements %IF and %ELSE in lines 13:19 do not generate any SAS semicolons; their only result is string literals for the title statement. The secondary effects – line 22 – are footnotes are cleared and page number is reset to one.

Further titles in a program may be placed after calling the TITLES macro using this example statement:
NOBS: number of observations of a data set, two utilities used in the FREQ examples:

+ 1 + 2 + 3 + 4 + 5 + 6 + 7
123456789012345678901234567890123456789012345678901234567890
01 /* iNOBS: Include Nobs 01
02 parameters: %GLOBAL LIBRARY DATA_SET
03 from: SAS Guide to Macro Processing, V6, 2nd ed., pg 263 03
04 note: call symput + RUN; NOBS has no value until loaded at step boundary 04
05 returns %GLOBAL macro-var NOBS: number of observations;*/ . . . . . . /* 05
06 DATA _NULL_; 06
07 if 0 then set &LIBRARY..&DATA_SET. nobs = Count; 07
08 call symput('NOBS',compress(put(Count,32.))); stop; 08
09 run;%PUT @@iNobs: &DATA_SET. has <&NOBS.> observations; 09

Remember what I said about using the pair of statements: call symput+run? Examine this program closely and understand not only what but also when NOBS is happening. As a test insert this line between line 08 and 09:

%put NOBS<&NOBS.>

You'll receive this message:

WARNING: Apparent symbolic reference NOBS not resolved.

because NOBS is uninitialized until the second statement of the pair in line 09.

Note snake eyes – double dots – in line 07. Remember that macro variables are delimited with a dot as suffix; the second dot is the infix in a two-level data set name.

%MACRO NOBS(_MAC_VAR /* macro-var name default=NOBS */
,DATA =./* data set name default=&SYSLAST.*/
,_GLOBAL_=0/* return %GLOBAL mac-var? default=%LOCAL */
);%LOCAL DSN; run; 10
11 %IF "&_MAC_VAR" = "" %THEN %LET _MAC_VAR = NOBS; 11
12 %IF &_GLOBAL_ %THEN %DO; %GLOBAL &_MAC_VAR.; %END; 12
13 %IF "&_DATA." = "." %THEN %LET DSN = &SYSLAST.;%*resolve mac-var; 13
14 %ELSE %LET DSN = &DATA.; 14
15 %LET DSID = %sysfunc(open(&DSN.)); 15
16 %IF &DSID %THEN %DO; %LET &_MAC_VAR. = %sysfunc(attrn(&DSID.,NLOBS)); 16
17 %LET RC = %sysfunc(close(&DSID.)); %END; 17
18 %ELSE %PUT Open for &DATA. failed:%sysfunc(sysmsg()); 18
19 %PUT NOBS: ";&_MAC_VAR."=<&&&_MAC_VAR.> data=&DSN.; %* . . .*NOBS; %MEND; 19
20 ;/*************************** to enable end this line with a slash (/) * 20
21 data X;doI=1 to 10;output;stop;run; 21
22 %NOBS();%PUT _USER_;%PUT NOBS=<&NOBS.>; 22
23 %MACRO TESTING(DATA); DATA &DATA.;do I = 1 to 12;output;stop;run; 23
24 %local NOBS_Y; 24
25 %NOBS(NOBS_Y);RUN;%PUT _LOCAL_;%*see DATA and NOBS_Y; 25
26 %MEND; 26
27 %TESTING(DATA1); 27
28 %PUT NOBS_Y=<&NOBS_Y.>;%*see error message: NOBS_Y does not exist; 28
29 ;="/*******************************TEST DATA closure*/ 29

The macro NOBS differs essentially from iNOBS in that it uses the set of system functions and does not end with a run statement. Since this is a fan-in macro, the calling macros declare a local macro variable in their macro symbol table, that is: in their scope. Each calling macro has a run statement after the call to NOBS, which allocates the macro variable within its scope. Examine the test data provided and note the call in FREQ1VAR, line 16.

WARNING: NOBS is useful in showing the style sheet in use: note how easy it is to scan the control statements on the left – LINES 11:18 --, then the conditionally executed statements in the center; and finally all closures on the right side, where they are not distracting.
Review this first program example and notice the two similar paragraphs. SAS-L's Ian Whitlock refers to this type of program as wallpaper: lots of repetition with little variation. Recognize the same library, same data set, same identifier, different variables, different formats and different titles.

W4R: Note the hanging indent of the options after the proc PRINT. Columns 11:16 contain the control statements for the procedure and its options. User-supplied information is placed in columns 18:72. This placement on the page is the key of the W4R style sheet: vertical alignment of keywords enables quick scanning to locate phrases or statements to examine and change.

With two examples in view, what is a template into which you can insert the common elements?

This template is derived from the FREQ program above. Lines 08:11 are the essential statements from lines 03:06 and 08:11 in the FREQ program. Note the documentation in lines 02:05; that is an important reminder of what you are doing.

Next we need a driver: a calling program to execute this as an %included file.

Call FREQ Basic is the driver that calls the simple FREQ program iFreqPrnt.

Review Autoexec Basic, line 09 to see the global macro variable LIBRARY. Review line 19, and see the option source2; this option enables the log to show line numbers of the %included files which is very important for debugging your statements!

Note the TITLES macro – called in line 02 – defines the global macro variable DATA_SET.

W4R: Control statements on the left:
1. TITLES which sets DATA_SET
2. VAR and FORMAT declarations.
Conditionally-executed %include statements in the center. When copying to a new project, the items to change – parameters to TITLES and VAR and FORMAT declarations – are aligned.
change happened!  and new requirements specifications need to be implemented.  IfFreqPrnt was a print routine; module IFreqSmry saves a summary data set with information in a standard data set structure—a summary object.

Note that the NOBS macro variables is the number of observations of the whole data set and that the where clause may exclude observations that are blank or invalid. The result could be that the sum of Count might not equal the number of observations.

W4R: note the where clause in lines 16:17, which is a data step option; it is placed in the central control column because it is a keyword. Likewise with the slash and out= options of the tables statement in lines 19:21, keywords in the center, user-supplied optional tokens to the right.

change has happened!  in this driver, we want to be able to switch between the old program IFreqPrnt, which produces only a print, and the new IFreqSmry which produces a summary data set.

Conditional execution of %include files: To enable the use of the print routine, disable line 03 with an asterisk:

*LET WHICH = SMRY;

This leaves the macro variable WHICH with the value of PRNT, which yields an include filename of (IFREQPRNT).

SAS2XLS is a short fan-in utility macro. Note in line 04 that PATH defaults to a global macro variable PATH_XLS, which is defined in a the LETPATH include file. This routine will not overwrite an existing file; lines 09:11 delete a previous file.
Converting %includes to macros:

Ok, so you’ve been writing for two years or have written ten thousand statements, whichever milestone came first for you. You are comfortable writing flawless paragraphs, even chapters, of SAS statements. You figure you’re ready to lose the line numbers, write macro statements in ALL CAPS, in order to remind yourself that you are switching gears between the number-crunching paradigm of SAS and the really simple string-handling of the macro language.

Let’s look at a conversion:

```sas
01 ;/*/ macro FREQ1VAR FREQ of One Variable 01
02 NOTE: need %GLOBAL mac-vars: BLANK INVALID 02
03 usage: 03
04 %FREQ1VAR(VAR1,FORMAT1); 04
05 %FREQ1VAR(VAR2,FORMAT2,DATA=P0001AD3); 05
06 %FREQ1VAR(VAR3,FORMAT3,LIBRARY=WORK); 06
07 output data set structure: N_Obs Label Value Count Percent;/*. . . */0 7
08 %macro FREQ1VAR(VAR /* variable name */ 08
09 ,FORMAT /* format with dot as suffix */ 09
10 ,LIBRARY = LIBRARY /* or LIBRARY=WORK */ 10
11 ,DATA = &DATA_SET./* NOTE: %GLOBAL mac-var */ 11
12 ,OUT = . /* default is &VAR */ 12
13 ,HTM_PATH = &PATH_HTM./* default to %GLOBAL mac-var */ 13
14 );%local NOBS;
15 15 %IF "&OUT." eq "." %THEN %LET OUT = &VARIABLE.;
16 16 %NOBS(data=&LIBRARY..&DATA.);run;
17 17 proc FREQ data = &LIBRARY..&DATA. 18
19 (where = (put(&VAR.,&FORMAT.) 19
20 not in ("&BLANK.",&INVALID."); 20
21 format &VAR. &FORMAT.; 21
22 tables &VAR. 22
23 / noprint 23
24 out = FREQ1VAR; 24
25 25 DATA &LIBRARY..&OUT.(label = ";LIBRARY..&OUT." 26
27 rename = (&VAR. = 27
28 %IF "%substr(&FORMAT.,1,1)" eq "%" %THEN %DO; ValueChr %END; 28
29 %ELSE %DO; ValueNum %END; 29
30 %* DATA closure; }); 30
31 attrib N_Obs length = 4 31
32 Label length =40 32
33 Count length = 4 label = "N=NOBS.";
34 retain N_Obs &NOBS.;
35 set FREQ1VAR;
36 Label = put(&VAR.,&FORMAT.);
37 37
38 %SAS2XLS(&OUT.); 38
39 39
40 ods listing close; /*SAS.list OFF; 40
41 ods html body = "&HTM_PATH.\&OUT..HTM";
42 proc PRINT data = &LIBRARY..&OUT.;
43 ods html close;
44 ods listing; /*SAS.list ON; 44
45 run;/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . %FREQ1VAR; %MEND; 45
```

The FREQ1VAR macro illustrates several conversion issues:

1. Parameter list: VAR and FORMAT are positional parameters because they follow the order of macro variable declarations in the previous calling programs.
   Named parameters: LIBRARY is hard coded to default to the name LIBRARY, but DATA defaults to the global macro variable DATA_SET, which is set by the fan-in macro TITLES.

Avoid this error: setting a local variable equal to a global variable of the same name: DATA_SET = &DATA_SET. This generates a circular reference, which leaves your session in an infinite loop!

2. local macro variables: The macro variable NOBS was global when generated by the include file iNOBS. In line 13 it is declared local in order to reduce the number of global macro variables. Note the run statement after the call of the macro NOBS; this step boundary in FREQ1VAR enables macro variable NOBS to be local to FREQ1VAR. If the step boundary were in the macro NOBS, the macro variable NOBS would be in the global symbol table.

   Similarly OUT has a default value of blank – line 12 – but is immediately tested and reset to the value of VAR when blank. Note the quotes around the values being compared. This is a mnemonic that reminds you the macro language is a string-processing language. Why use a default value of dot? Because you can see it! You want to avoid this test: %IF "&OUT = %THEN ...

   This expands to: %IF "&OUT. eq blank %THEN ...

   I recommend: %IF "&OUT." eq "dot" %THEN ... because it’s more readable.

3. %THEN %DO lines 28:29 ValueChr and ValueNum are the new names of the FREQ tables variable. This construct shows the macro language inserting the appropriate token in the data statement rename phrase which starts on line 27 and ends on line 30. Without the %DO .. %END bracket, there would be a semicolon after each of the two tokens; these are the closure of the %IF and %ELSE macro statements, not to be confused as closure of the data statement.

4. use of global macro variables as default values of parameters: DATA and PATH_HTM. Note that the macro SAS2XLS uses the global macro variable PATH_XLS.

This macro is a shortened version of Fehd (1999) %FREQ1VAR,
CONCLUSION
There are several important issues to consider when using macros:
* Maintenance: what is the skill level of the maintenance programmer?
* Global macro variable usage: where do changes occur? As an example, changing the value of the macro variable LIBRARY from LIBRARY to WORK, would cause all subsequent programs to abend with a ‘data set not found’ error message.
* Ensuring local macro variable usage
* Programming style: ad hoc, planned, or revised and revisited?

Here ends the tour of a decade-long process of converting programs with many repetitive paragraphs, to the simpler style of using includes. The simple modules developed are reusable. The include style is expensive in its use of global macro variables.

Moving from includes to macros is expensive in terms of development because of the moderate learning curve for the macro language. In the long run, macro usage is simpler, and less expensive in terms of global macro variable usage. In addition new project development is quicker because of reusable modules and the minimal changes necessary to customize copied programs.

REFERENCES