Five Easy (To Use) Macros

Ted Clay, Clay Software & Statistics, Ashland, OR

Consider increasing your SAS vocabulary by five words: %DIR, %MAKEFMT, %TRANSPO, %ARRAY and %DO_OVER. Macro definitions are in the appendix. %DIR creates a SAS data set with the list of files in a directory. %MAKEFMT creates a format from a data set. %TRANSPO changes data from a series of observations to one observation per by-group with arrays of variables, preserving variable attributes. Finally, %DO_OVER, with the support of its companion %ARRAY, generates repeated program code. %DO_OVER loops over a list of values, substituting them wherever you put a "?" in your code phrase. For example, RENAME %DO_OVER(values=A B C, phrase= ?=?suf); generates the statement RENAME A=Asuf B=Bsuf C=Csuf; The code phrase can consist of many statements or an external macro, and multiple arrays can be defined and looped-over in parallel. No other macros have had a greater impact than these two, appearing in over 40% of the author's statistical programs.

OVERVIEW

This paper is about making use of some handy macros. Here are one-line descriptions of the macros:
%DIR creates a data set with the list of files found in a directory.
%MAKEFMT creates a format from a data set.
%TRANSPO restructures data to have one observation per by-group, preserving variable attributes.
%DO_OVER generates repeated program code, with the assistance of its companion macro …
%ARRAY which stores text in an array of macro variables.

In order to spend more time on real-world examples of their use, this paper treats these macros as "black boxes." The complete source code is attached as an appendix. A brief paragraph near the end of the paper explains how you would use the SASAUTOS option to make them automatically available in your programs. Two papers explaining %TRANSPO and %DO_OVER in more depth, are mentioned in the References section at the end.

THE %DIR MACRO

THE NEED

This little macro gets its name from the DOS command "dir" which shows you a list of files in a directory. What you get from the "dir" command or its equivalent can vary from system to system, and so over time the need for a system-independent version of this macro became clear. We employed some built-in SAS functions to obtain information directly from the operating system.

THE MACRO

The first parameter is positional, and is the name of the directory you want to examine, relative to the current directory. The %DIR macro produces a SAS data set with the default name "DIR.". For example,

```
%DIR(macros)
proc print data=dir;
run;
```

Produces:

Obs	Directory	FileName	Short Name	Extension
1	c:\claysoft\presentations\pnwsug2006\macros	ARRAY.sas	ARRAY	sas
2	c:\claysoft\presentations\pnwsug2006\macros	DIR.sas	DIR	sas
3	c:\claysoft\presentations\pnwsug2006\macros	DO_OVER.SAS	DO_OVER	SAS
4	c:\claysoft\presentations\pnwsug2006\macros	MAKEFMT.SAS	MAKEFMT	SAS
5	c:\claysoft\presentations\pnwsug2006\macros	TRANSPO.sas	TRANSPO	sas

THE %MAKEFMT MACRO

THE NEED

Formats can be tremendously helpful as an efficient lookup table. The efficiency comes partly because the format is loaded into memory, resulting in no disk access when you need to look up a value. It also comes in the form of a simpler program, with no sorting and merging to accomplish the same thing. Unless your data is huge, the second type of efficiency will be the most important to you. Using the %MAKEFMT macro, table lookup becomes a two-line affair, one to create the format, one to use it.

You may already use the CNTLIN= option of PROC FORMAT, which allows you to create a format from variables in a data set. There are certain shortcomings which the %MAKEFMT macro takes care of. What is the normal result when you use the format on a value which is not contained in the format? You get the original unformatted value. In most real-world programming situations, you want to know when this has happened, and sometimes it can be difficult to distinguish between formatted and unformatted values. The solution is to add an "OTHER" case to the format. This may be easy enough to do when you are writing out a VALUE statement, but awkward when you are creating a data set for use with CNTLIN. It was always just enough hassle that this SAS programmer never bothered with it until building it into a macro.

THE MACRO

The %MAKEFMT macro automatically adds an observation to the data set to format any "OTHER" cases to blanks or a text string you specify. Missing values can be given their own label even if there are no missing values in the data set. It also allows you to assign a constant text value to the label, instead of a variable value. By handling these special situations, the macro makes the CNTLIN= feature of PROC FORMAT a hassle-free tool. The parameters of the %MAKEFMT macro include:

FMTNAME = Name of format to be created.
DATA = Name of input data set.
VALUES= The character or numeric variable containing values to be formatted.
LABELS= A variable containing the value labels, or "<string>" in quotes. If it is a string in quotes, all values in the data set will have the same label.
OTHER = (optional) Text label for the "other" case. The default is blank.
MISSING= (optional) Text label for missing values.

EXAMPLE:

%MAKEFMT to check validity. Suppose you have a list of valid values, and want to detect any invalid values in your main data set. The following code would quickly select the observations with an invalid ICD9 code in the Primary Diagnosis variable:

```
%MAKEFMT(FORMAT=$icd9fmt,DATA=ICD9,VALUES=code,LABELS=descrip,OTHER=Invalid);
DATA invalids;
   set main (keep=PrimaryDX PatientID)
   if put(PrimaryDX,$icd9fmt.) = 'Invalid';
run;
```

THE %TRANSPO MACRO

THE NEED

You are trying to manage your data. Your data is running down the page, but you really want it to run across the page. Your dataset is long and skinny, and you want it to be short and fat. If you only have one variable to deal with, no problem. PROC TRANSPOSE with only one variable on the VAR statement will do the trick nicely. But this may not be the case. When you want to transpose more than one variable, PROC TRANSPOSE will produce a separate observation for each variable (within each by-group). Often what you really want is to have a single observation per by-group with all the variables on it. And you would like the new variables to be given meaningful names related to your original names. The second limitation with PROC TRANSPOSE is that you cannot transpose a mixture of numeric and character variables. Yes, PROC TRANSPOSE has an "answer" to this situation – it converts all the numeric variables to character – but I have yet to meet anyone who actually wanted this to happen. Finally, PROC TRANSPOSE leaves all your variable attributes behind. Any formats, labels and lengths which you had on your input variables get lost. Formats and labels are made blank, and the lengths of numeric variables all become 8. This is a necessary consequence of the way PROC TRANSPOSE "flips" a matrix of variables and observations.

THE MACRO

By trying to **do less**, we are able to **preserve more**. The %TRANSPO macro outputs all variables onto a single observation per by-group. For example, if you have a dataset with variables A, B and C and up to 4 observations per by-group, the output will be one observation per by-group with variables A1-A4, B1-B4 and C1-C4. The variables can be a mixture of character and numeric, and their lengths, formats and labels carry over to the corresponding output variables. The suffixes can be determined by an ID variable.

Here are the parameters of %TRANSPO:
DATA = Name of input dataset.
OUT = Name for output dataset.
BY = By-grouping variable(s). Output data set will have one observation per unique BY-value.
VARS = Variables to be transposed. Can be any combination of character or numeric.
ID = (optional) Name of variable whose (formatted) value will determine the suffix of the new variable names. If no ID variable is given, the suffixes of the new variables will be the numbers 1,2,3, etc.
VARSEP = (optional) Text to insert between the old variable name and the variable suffix.
LABELSEP = (optional) Text to insert between the old label and the variable suffix.

The output variable names consist of (1) the original variable name, concatenated with (2) the optional VARSEP text string, concatenated with (3) the formatted values of the ID variable, or else "1", "2", "3" if no ID variable is given. (I have found that the ID variable is needed 95% of the time.)

EXAMPLE

```
%TRANSPO(DATA=officers,OUT=clubs,
BY=club,
ID=posit,
VARS=name dob,
VARSEP=_,           /* Notice that the output variables have underscores  */
LABELSEP=of);       /* Notice that the word "of" appears in output labels */
```

INPUT DATASET: OFFICERS

Club	Posit	Name	DOB
Poker	Pres	Jack	04/03/1970
Poker	VP	Fred	09/24/1981
Poker	Treas	Beth	07/19/1921
Yoyo	Pres	Joan	02/09/1988
Yoyo	VP	Hal	12/25/1970

OUTPUT DATASET: CLUBS

Club	Name_Pres	DOB_Pres	Name_Treas	DOB_Treas	Name_VP	DOB_VP
Poker	Jack	04/03/1970	Beth	07/19/1921	Fred	09/24/1981
Yoyo	Joan	02/09/1988			Hal	12/25/1970

Notice that the Yoyo club has no Treasurer, so the Name_Treas and DOB_Treas variables are missing.

INPUT DATA SET CONTENTS

```
Variable Typ Len Format Label
-----------------------------------
Club      C 7           Club
Posit     C 5           Position
Name      C 4           First Name
DOB       N 8 MMDDYY8.  Date of Birth
```

OUTPUT DATA SET CONTENTS

```
Variable Typ Len Format Label
-----------------------------------
Club       C 7          Club
Name_Pres  C 4          First Name of Pres
DOB_Pres   N 8 MMDDYY8. Date of Birth of Pres
Name_Treas C 4          First Name of Treas
DOB_Treas  N 8 MMDDYY8. Date of Birth of Treas
Name_VP    C 4          First Name of VP
DOB_VP     N 8 MMDDYY8. Date of Birth of VP
```

Notice that the date format of DOB carried over to all three DOB variables in the output.

THE %ARRAY AND %DO_OVER MACROS

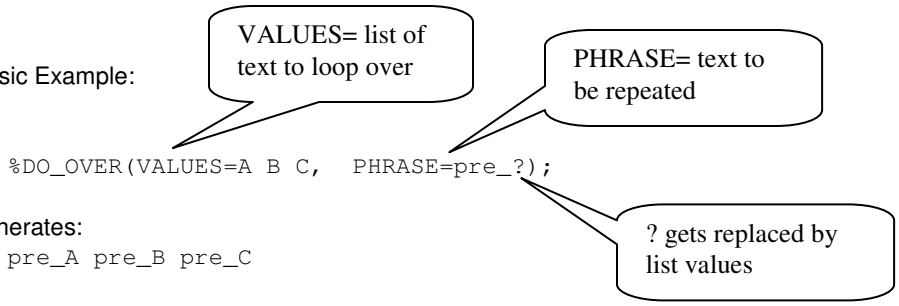
THE NEED

Macro %DO-looping is common practice in many programs using the macro language. Unfortunately, the usual solutions involve somewhat awkward macro syntax, with double ampersands and macro definitions which can break up the continuity of programs. The %ARRAY and %DO_OVER macros allow you to hide the repetitive machinery, resulting in programs that are shorter and more readable. Because these macros are self-contained and use global macro variables, you can use them freely in "open code". The macros use regular characters as much as possible, freeing you from the need for macro quoting functions.

%DO_OVER BASICS

The %ARRAY and %DO_OVER macros are analogous to the ARRAY and DO OVER statements in the SAS® data step language, which define and loop over implicitly subscripted arrays. In the data step language, you must always have an ARRAY statement before you can use DO OVER. But the %DO_OVER macro with a VALUES= parameter is self-sufficient. We start here.

Basic Example:



generates:

```
pre_A pre_B pre_C
```

Table 1: Applications of %DO_OVER with different PHRASE parameters

Application	Code Example	Generates
1. Bulk renaming	`rename %DO_OVER(VALUES=A B C, PHRASE=?=pre_?) ;`	`rename A=pre_A B=pre_B C=pre_C;`
2. A quoted list	`if letter in ( %DO_OVER(VALUES=A B C, PHRASE="?"));`	`if letter in ("A" "B" "C");`
3. Tracking merged data	`Merge %DO_OVER(VALUES=A B C, PHRASE=?(in=in?));`	`merge A(in=inA) B(in=inB) C(in=inC);`
4. Complete statements	`Proc freq; %DO_OVER(VALUES=A B C, PHRASE=table ? / out=freqs?;)`	`Proc freq; table A / out=freqsA; table B / out=freqsB; table C / out=freqsC;`
5. Multiple statements	`%DO_OVER(VALUES=A B, PHRASE= title "Printout of ?"; proc print data = ?;)`	`Title "Printout of A"; Proc print data = A; Title "Printout of B"; Proc print data = B;`
6. Macro language uses	`%LET OLD=A B C; %LET NEW=%DO_OVER(VALUES=&OLD, PHRASE=pct_?);`	`Pct_A pct_B pct_C (assigned to variable NEW)`
7. The default phrase (single question-mark)	`%DO_OVER(VALUES=A B C, PHRASE=?) %DO_OVER(VALUES=A B C);`	`A B C A B C (the same result)`

DISCUSSION

Close parentheses, and correct placement of semi-colons, make a big difference. The %DO_OVER macro begins with an open parenthesis, and needs a close parenthesis. In parsing the PHRASE= parameter, the macro processor continues until it finds a comma or an unbalanced close parenthesis. Items 2 and 3 above both end with the same three characters: "));" but they have very different meanings. In Item 2, the %DO_OVER macro was inside a pair of parentheses, so the first final close-parenthesis marks the end of the PHRASE parameter, and the second close-parenthesis and semicolon are outside the macro. In Item 3, the PHRASE contains an expression in parenthesis: "(in=in?)". It is immediately followed by %DO_OVER's closing parenthesis and the semi-colon ending the MERGE statement. In Item 4, the semi-colon is inside the macro close parenthesis. The PHRASE contains a complete SAS statement which gets repeated. In Item 5, two complete statements get repeated. Inside %DO_OVER is machinery to parse the VALUES= string into discrete items of a list, store them in a internal hidden macro array, and substitute those values in the phrase. Next we will examine macro arrays.

THE MACRO ARRAY STRUCTURE

A macro array is a list of macro variables sharing the same prefix and a numerical suffix. The suffix numbers run from 1 up to a highest number. The value of this highest number, or the length of the array, is stored in a macro variable with the same prefix, plus the letter "N". The prefix is also referred to as the name of the macro array.

Example: The macro array "DAYS" containing the first three days of the work-week.

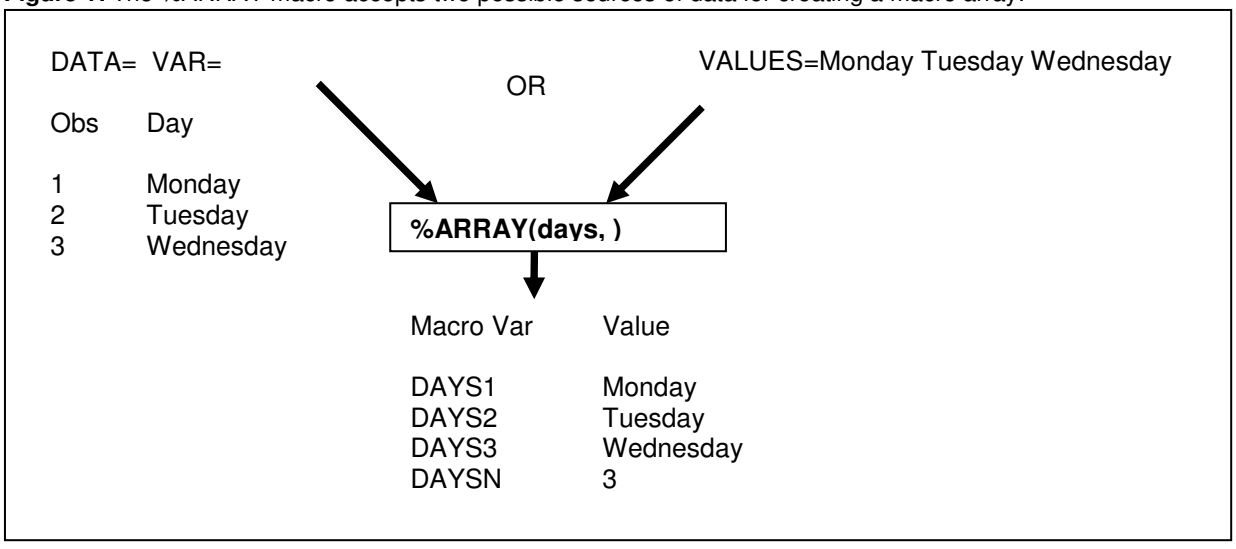
Macro Variable Name	Contents
DAYS1	Monday
DAYS2	Tuesday
DAYS3	Wednesday
DAYSN	3

The secret to the power of this structure is in the last row By also storing the length of the array, using a macro variable name with the same prefix and a standard, predictable suffix, you only need to give the prefix or name of the array, in this case "DAYS." Note that it does not use a macro variable called "DAYS" -- a macro variable and macro array of the same name can coexist.

CREATING AND USING A MACRO ARRAY

The purpose of the %ARRAY macro is to create a macro array and load text into it from either a SAS® data set or an explicit list of values.

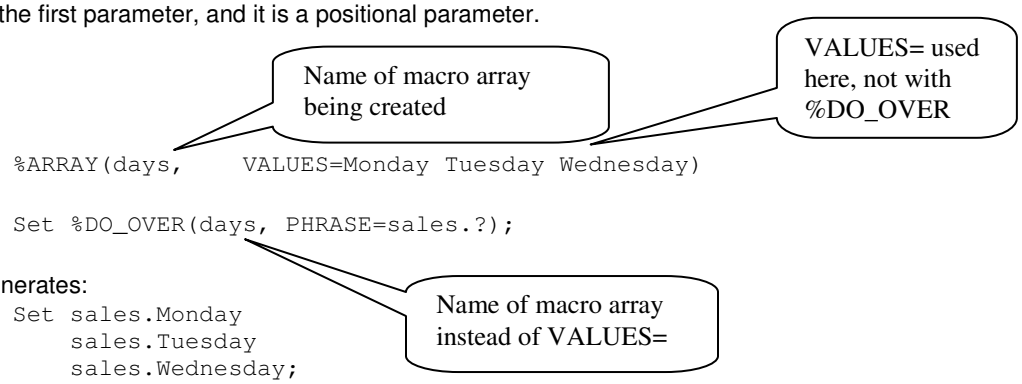
Figure 1: The %ARRAY macro accepts two possible sources of data for creating a macro array:



Why create a macro array? (1) You want to use a list in many places, (2) You like the "sound" of the abbreviated syntax. (3) It is a quick way to get variable values from a data set into the macro environment.

%ARRAY WITH VALUES=

We use the familiar VALUES= parameter, but this time to create a macro array. The macro array named "days" is used in the %DO_OVER macro. In both %ARRAY and %DO_OVER the name of the macro array is the first parameter, and it is a positional parameter.

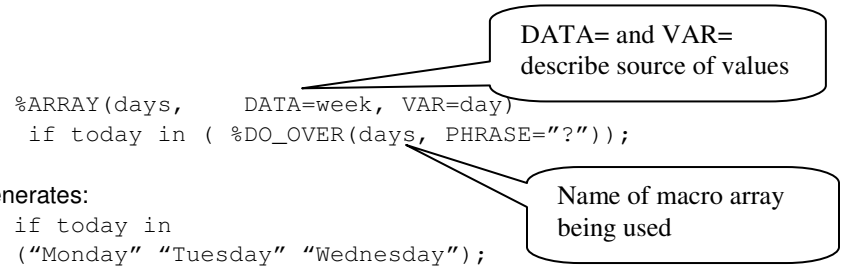


generates:

```
Set sales.Monday
    sales.Tuesday
    sales.Wednesday;
```

%ARRAY WITH DATA= AND VAR=

Often you need to create a macro array using data already stored in a data set. Suppose we have a data set "week" with a variable "day" as in the above graphic. The following code shows how you would (1) assign values from the data set into a macro array, and (2) use the macro array with %DO_OVER.



generates:

```
if today in
("Monday" "Tuesday" "Wednesday");
```

SCOPE AND PLACEMENT CONSIDERATIONS

Macro array variables are declared global, so they are available throughout a program after they are created. The %ARRAY macro with DATA= must be outside any DATA or PROC step. So in fact the two statements in the preceding example could not be right next to each other. No other restrictions apply to the placement of %ARRAY or %DO_OVER. They can be located in the tight space of a single statement in open code. %DO_OVER can be nested.

PERSPECTIVE

The concept of macro arrays closes an important gap in the SAS® system. The %ARRAY macro strongly links the world of data to the world of macro variable values. Once defined, you do not have to be concerned with how many elements a macro array contains, only referring to the macro array by its name. %DO_OVER allows you to execute SAS code plugging in the values from macro arrays. Both are designed to avoid the necessity of the macro language ampersand, or special quoting considerations. These macros extend the power and scope of the SAS programming language.

I took a survey of the non-trivial programs developed for a recent project, eliminating programs that were nearly identical to other programs. I found that of 22 programs, 11 made use of %DO_OVER. It is difficult to name another feature of the SAS® system, with the exception of PROC SORT, which has had such a major impact on the author's programming style.

The following table illustrates the range of additional features.

FULL FEATURES OF %ARRAY AND %DO_OVER

Table 2: Significant Additional Features.
In these examples assume that we have already created a macro array named "ABC" with values "A", "B" and "C". We would do this by `%ARRAY(abc,VALUES=A B C);`

Feature	Code Example	Generates
1. VALUES= a numeric list	`%ARRAY(yrs,VALUES=1983-1986);` `%PUT %DO_OVER(yrs);`	`1983 1984 1985 1986`

	`%DO_OVER(VALUES=time4-time7)`
	`%DO_OVER(VALUES=4-7,`
	`PHRASE=time?)`
		`time4 time5 time6 time7`
		
		`time4 time5 time6 time7`

2. Handling imbedded blanks	`%DO_OVER(VALUES=`
	`Alameda/San Mateo/Santa Clara,`
	`DELIM=/,PHRASE="?")`
		`"Alameda"`
		`"San Mateo"`
		`"Santa Clara"`

3. Inserting something between values	`%DO_OVER(abc,`
	`PHRASE=if letter="?" then ?=1;,`
	`BETWEEEN=else)`
		`if letter="A" then A=1;`
		`else if letter="B" then B=1;`
		`else if letter="C" then C=1;`

3b. The BETWEEN= COMMA keyword	`maxabc = max(of %DO_OVER(abc,` `BETWEEN=COMMA));`	`Maxabc = max(of A,B,C);`

4. Inserting the array index using "?_i_"	`Merge %DO_OVER(abc,`
	`PHRASE=?(in=in?_i_));`
		`Merge A (in=in1)`
		`B (in=in2)`
		`C (in=in3);`

5. Selecting a subset of data for an array	`%ARRAY(males,`
	`DATA=subjects(where=(sex='M')),`
	`VAR=casenum)`
		`* Puts only the male subjects'`
		`casenums into a macro array;`

6. Creating multiple macro arrays from data	`%ARRAY(Xs Ys,` `DATA=temp,VAR=X Y)`	`* Variable X into array Xs;` `* Variable Y into array Ys;`

7. Looping over multiple macro arrays	`%ARRAY(abc,VALUES=A B C);`
	`%ARRAY(def,VALUES=D E F);`
	`Rename %DO_OVER(abc def,`
	`PHRASE=?abc=?def);`
		`Rename A=D B=E C=F;`

8. Passing array values to a macro	`%MACRO doit(xxx);`
	`<code involving &XXX>`
	`%MEND;`
	`%DO_OVER(abc,MACRO=doit)`
		`%doit(A)`
		`%doit(B)`
		`%doit(C)`

9. Passing multiple array values to a macro	`%MACRO doit(xxx,yyy);`
	`<code involving &XXX and &YYY>`
	`%MEND;`
	`%DO_OVER(abc def,MACRO=doit)`
		`%doit(A,D)`
		`%doit(B,E)`
		`%doit(C,F)`

Feature 8, the MACRO= parameter, is used in "REAL WORLD APPLICATION #3" below.

REAL WORLD APPLICATION #1

THE TASK:

Several projects needed information about regions of the state of California. Too many phone conversations began with the phrase "Where is the format for …?" We needed a single repository of information about regions, from which formats could be derived. We wanted a solution that would be easy to understand and to easy to maintain.

THE SOLUTION

We created a data set with one observation per region, with all the pertinent information. (How you choose to create and maintain that data set may vary.) Then the %MAKEFMT macro created a set of formats to look up characteristics of each geographical entity. In this example, we use one of these formats to create a new variable "region" in our data file. References to format libraries are removed for simplicity.

```
* Table with all geographic entities;
data geogs;
   input GEOG 5. +1 GEOGC $5. +1 REGION $4. +1 GEOGABB $4. +1 GEODESC $40.;
   label
     GEOG    = 'Geographic Area (N)'
     GEOGC   = 'Geographic Area (C)'
     REGION  = 'Geographic Region (C)'
     GEOGABB = 'Geographic Area Abbreviation'
     GEODESC = 'Geographic Area Description' ;
datalines;
0     00         00CA  0 California
0.2   00.2       00BA  0.2 Bay Area
0.3   00.3       00SJ  0.3 San Joaquin Valley
0.4   00.4       00SE  0.4 Southeast Counties
0.7   00.7       00SA  0.7 Sacramento Area
0.8   00.8       00CC  0.8 Central Coast
0.9   00.9       00NM  0.9 North-Mountain
1     01    00.2 01AM  1 Alameda
1.1   01.1       01BK  1.1 Berkeley
2     02    00.9 02AL  2 Alpine
3     03    00.9 03AD  3 Amador
4     04    00.9 04BU  4 Butte
5     05    00.9 05CV  5 Calaveras
6     06    00.9 06CU  6 Colusa
7     07    00.2 07CC  7 Contra Costa
8     08    00.9 08DN  8 Del Norte
9     09    00.7 09ED  9 El Dorado
                                  < lines removed >
58    58    00.7 58YU 58 Yuba
99    99         99US 99 Federal
;;;;

%MAKEFMT(data = geogs, values = GEOG, labels = GEODESC, fmtname = GEOG);
%MAKEFMT(data = geogs, values = GEOGC,labels = GEODESC, fmtname = $GEOG);
%MAKEFMT(data = geogs, values = GEOG ,labels = GEOGABB, fmtname = GEOGABB);
%MAKEFMT(data = geogs, values = GEOG ,labels = REGION,  fmtname = GEOGREG);

* Create variable "Region" from "Geog" using the geog-to-region format;
DATA temp;
  SET saved.stats;
  length Region 8;               * Create numeric variable Region;
  Region = put(geog,geogreg.);   * Look up the region using GEOGREG format;
Run;
```

REAL WORLD APPLICATION #2

THE TASK:

The county health officials are concerned about change over time in the rates of various health problems in their county. We are preparing "databooks" for them displaying the last 12 years of data. The data is currently structured with variables: County, Indicator, Year, Num and Den (numerator and denominator). Year contains values 1993, 1994, 1995, etc. For reporting, and to apply certain statistical methods, we need to restructure the data so that for each county and indicator we have "Num1993" "Den1993" "Num1994", "Den1994" etc.

THE SOLUTION

```
%TRANSPO(
     DATA=saved.CountyLong,
     BY=county indicator,
     VAR=num den,
     ID=year,
     OUT=saved.CountyWide)
```

REAL WORLD APPLICATION #3

THE TASK:

More than 100 Access databases have been uploaded from various hospitals to a directory (c:\project1\access). Each contains the same set of 6 tables with the same structure but different data. We need to create 6 SAS® data sets concatenating the corresponding tables from all the Access databases.

THE SOLUTION

```
* Create macro array with list of Access databases (a.k.a MDB files);
%DIR(c:\project1\access);
%ARRAY(mdbs,DATA=dir,VAR=filename);

* Create macro array with list of tables found inside each one;
%ARRAY(tbls,VALUES=HospitalInfo     HospitalAdmissions  ICUAdmissions
                   InfectionEvents  ProcessMeasures     Outcomes)

* Define a macro to read all tables from a single Access database;
%MACRO READIT(mdbfile);
   * Process each table in the list;
   %DO_OVER(tbls,phrase=
         * Import the table;
         PROC IMPORT dbms="access" table="?" out=? replace;
           database="c:\project1\access\&MDBFILE";
         run;

         * append the table to the SAS data set of the same name;
         PROC APPEND base=saved.? data=?;
         run;
      ) * close-parenthesis of DO_OVER;
%MEND;

* Execute the above macro for each Access database in the macro array "mdbs";
%DO_OVER(mdbs, macro=READIT);
```

REAL WORLD APPLICATION #4

THE TASK:

"Put each hospital's patient list into a separate sheet in an Excel workbook." We have a data set called PatientData with a variable Hospital_ID plus other variables containing patient information. We want to make the data accessible by putting it in an Excel workbook, putting each hospital on its own sheet.

THE SOLUTION

```
* Obtain a data set with the list of Hospital_ID values in the data;
PROC FREQ data=saved.PatientData;
   table Hospital_ID / noprint out=hospids;
Run;

* Create a macro array "HHH" containing the hospital IDs;
%ARRAY(HHH,data=hospids,var=Hospital_ID);

* Within DO_OVER, use the "where=" option to select each hospital;
%DO_OVER(HHH,phrase=
    PROC EXPORT data=saved.PatientData(where=(Hospital_ID="?"))
                file="Exported.xls";
       sheet="?";
    Run;
  ) * close-parenthesis of DO_OVER;
```

Modifying the PHRASE= parameter, you can perform **any** process as if you were using a "BY" statement.

MAKING IT WORK FOR YOU

If you haven't already, create a directory where you keep your general-purpose macros. For example you might use 'C:\SASMACROS\GENERAL'. Copy these macros and any other general-purpose macros into this directory. Make sure your programs always start with statements that include the following:

```
OPTIONS SASAUTOS = ('C:\SASMACROS\GENERAL'  SASAUTOS);
```

This tells SAS® where to look for a macro which is not defined in your program. SAS® assumes macros are stored in files with the same name as the macro, and with the extension ".sas". First it will look in your macro library, and next it will look in "SASAUTOS" which is the name of a directory where the SAS® system stores additional macro definitions. If you use an "AUTOEXEC.SAS" file, which contains code which is executed when SAS starts, it is a good idea to put the OPTIONS SASAUTOS statement there, rather than in each program.

CONCLUSION

The %DIR and %MAKEFMT macros improve the utility of standard features of SAS, lowering the hassle threshold involved in getting a list of files or making a format from data. The %TRANSPO macro implements a "better transpose" which preserves variable attributes, a result that normally requires extensive programming effort. Finally, %DO_OVER and %ARRAY can easily reduce the size and complexity of many data manipulation and analysis programs.

One way to perceive your own progress as a SAS programmer, is not only the i**nvisible** knowledge inside your head, but also a **visible** collection of useful tools Make your own collection of useful general-purpose macros, and consider adding to it the macros discussed in this paper.

ACKNOWLEDGEMENTS

David Katz, David Katz Consulting, for the macro array concept and its use with an externally defined macro. And to the many other macro programmers who have created and used macro arrays, written a macro to read a directory, or written a macro to create a format from data.
Art Carpenter, California Occidental Consultants, for aid in coding at the 2003 WUSS Code Clinic.

REFERENCES

"Tight Looping With Macro Arrays" SUGI Proceeding, 2006
"A Better Transpose: %TRANSPO Macro Preserves Variable Attributes" WUSS Proceedings, 2003

ABOUT THE AUTHOR

Ted Clay, M.S. is a statistical consultant and data analyst. His clients have included pharmaceutical companies, manufacturing companies, and grass-roots organizations, as well as research projects in epidemiology and health policy at the University of California San Francisco.

Your comments and questions are valued and encouraged. Contact the author at:

Ted Clay
Clay Software & Statistics
168 Meade St.
Ashland, OR 97520
Work Phone: 541-482-6435
Fax: Same
Email: tclay@ashlandhome.net

Appendix

SOURCE CODE FOR %DIR

```
%MACRO DIR(path,out=DIR);
 /* last modified 8/4/2006
                                                                  72nd col -->|
  Function:
     Create a data set with names of files found in a given directory

  Parameters:
    PATH      = Name of directory (positional).
                 May be relative to current directory.
    OUT       = Name for output data set

    Output dataset will have the following variables:
       Directory - Name of directory.
                  Full path not relative to current directory.
       FileName  - Full File name with extension.
       ShortName - First part of name, before the final dot.
       Extension - Part of full file name after the final dot.

  Author:  Ted Clay, M.S.
           Clay Software & Statistics
           tclay@ashlandhome.net  (541) 482-6435
        "Please keep, use and pass on, with this authorship note.
                    -Thanks "
          Send any improvements, fixes or comments to Ted Clay.
 */

%if %length(&PATH)=0 %then %let PATH=.;

data &OUT;
 length Directory $100 FileName ShortName $60 Extension $16;
 length optname $12 ;
 rc = filename("mydir","&PATH");
 did = dopen("mydir");
 if did > 0 then
 do;
   infocnt = doptnum(did);
    put infocnt=;
   do opt=1 to infocnt;
      optname = doptname(did,opt);
      if left(optname)='Directory' then Directory = dinfo(did,optname);
   end;
 filecnt = dnum(did);
 do f = 1 to filecnt;
    FileName  = dread(did,f);
    if index(filename,'.')>0 then
      do;
        lastdotcol = length(trim(filename))
                     - index(left(reverse(filename)),'.') +1;
        ShortName  = substr(filename,1,lastdotcol-1);
        Extension  = substr(filename,  lastdotcol+1);
      end;
    else do;
        ShortName  = filename;
        Extension  = ' ';
      end;
    output;
```

```
 end;
 end;
 else put 'ERROR: DIR macro was given an invalid path';
 keep Directory FileName ShortName Extension;
 label Directory = 'Complete directory path'
       FileName  = 'File name with extension'
       Shortname = 'File name without extension'
       Extension = 'Extension (after final dot)';
run;

%MEND;
```

SOURCE CODE FOR %MAKEFMT

```
%MACRO MAKEFMT(data=,values=,labels=,fmtname=,library=WORK,
               other=%str( ),missing=,leftvar=,rightvar=);

  /*  last modified 8/4/2006
                                                              72nd col -->|
  Function: Make a format from variables in a dataset

  Author:
     Ted Clay
     Clay Software & Statistics
     1-541-482-6435
     tclay@ashlandhome.net

  Parameters:
     DATA     -- Name of input dataset (required)
     VALUES   -- Name of variable containing the values to be formatted
                 (required)
     LABELS   -- Name of variable containing the formatted text, or a
                 character string in quotes (required). If a quoted
                 string is given it becomes the label of all values
                 in the data set. If LABELS a numeric variable, these
                 are converted to a character string using its associated
                 format, if any.
     LIBRARY -- Name of library where format is to be stored.
                 (default=WORK)
     FMTNAME -- Name of format (required).  It is as it would appear on
                 a VALUE statement, without a period) Max 8 characters
                 long counting the "$".  If VALUES is a character
                 variable, FMTNAME must begin with a "$".
     OTHER    -- A label to use if value is not found. (Default is blank).
                 Quotes are optional.
     MISSING -- A label for missing value.  If supplied it overrides
                 any label for missing values already specified in the
                 data set.  Quotes are optional.

   Parameters maintained for backward compatibility:
     LEFTVAR  -- obsolete name of VALUES parameter
     RIGHTVAR -- obsolete name of LABELS parameter.

  */

%local ERRORS Lfound Vfound labelstring;

%* Copy obsolete parameters into the current parameters;
%IF %LENGTH(&VALUES)=0 %then %let VALUES=&LEFTVAR;
%IF %LENGTH(&LABELS)=0 %then %let LABELS=&RIGHTVAR;

* Find out what the Type of VALUES;
```

```
proc contents data=&DATA noprint out=conts;
run;

%LET VFOUND=NO;
%LET LFOUND=NO;
%LET ERRORS=NO;

%LET VALUES = %UPCASE(&VALUES);

%let LABELSTRING=NO;

%if %QUOTE(%sysfunc(dequote(&LABELS))) NE %QUOTE(&LABELS)
 %then %let LABELSTRING=YES;

%IF &LABELSTRING=YES %then %let LFOUND=YES;
%ELSE %LET LABELS = %UPCASE(&LABELS);

%IF %LENGTH(%STR(&OTHER))>0 %then
                %LET OTHER  =%sysfunc(dequote(&OTHER));
%IF %LENGTH(%STR(&MISSING))>0 %then
                %LET MISSING=%sysfunc(dequote(&MISSING));

data _null_;
 set conts;

 if upcase(name) = "&VALUES" then
  do;
    call symput('VFOUND','YES');
    if type=1 then call symput('TYPE','NUM');
    else           call symput('TYPE','CHAR');
  end;

 %IF &LFOUND=NO %then
 %do;
 if upcase(name) = "&LABELS" then
  do;
    call symput('LFOUND','YES');
  end;
 %end;

run;

%IF &VFOUND=NO %THEN
   %DO;
       %PUT ************************************************************;
       %PUT ERROR IN MAKEFMT: Variable &VALUES not on input dataset;
       %PUT ************************************************************;
       %LET ERRORS=YES;
   %END;

%IF &LFOUND=NO %THEN
   %DO;
       %PUT ************************************************************;
       %PUT ERROR IN MAKEFMT: Variable &LABELS not on input dataset;
       %PUT ************************************************************;
       %LET ERRORS=YES;
   %END;

%PUT type is &TYPE;

* Check that $ is first char of FORMAT if VALUES is character.;
%IF %SUBSTR(&FMTNAME,1,1)=$ AND &TYPE=NUM %THEN
   %DO;
%PUT ****************************************************************;
```

```
%PUT ERROR IN MAKEFMT: &VALUES is numeric but format begins with "$";
%PUT ****************************************************************;
       %LET ERRORS=YES;
   %END;
%IF %SUBSTR(&FMTNAME,1,1) NE $ AND &TYPE=CHAR %THEN
   %DO;
%PUT ****************************************************************;
%PUT ERROR IN MAKEFMT: &VALUES is char but fmt does not begin with "$";
%PUT ****************************************************************;
       %LET ERRORS=YES;
   %END;

%IF &ERRORS=NO %THEN
%DO;

data xxxx;
 set &DATA end=lastobs;
 length FMTNAME $8;
 if _n_=1 then foundmissing=0;
 retain foundmissing;
 FMTNAME="&FMTNAME";
 %IF &LABELSTRING=NO %then
 %do;
    format  &LABELS; * remove format so that next statement will work;
 %end;

 %* This handles situation where data already contains variable "label"
    with length already set;
 %IF %quote(&LABELS) NE LABEL %THEN %DO;
    length label $100;
    label = &LABELS; * possible conversion from numeric to char here;
   %END;

 if missing(&VALUES) then
    do;
        foundmissing=1;
        %IF %LENGTH(&MISSING)>0 %then label = "&MISSING" %str(;);
    end;
 output;

 if lastobs then
  do;
      %* Set the value to missing;
      %IF &TYPE=NUM %then &VALUES=.%str(;);
      %ELSE &VALUES=' '%str(;);

      %* If the missing value was alredy found in the data,
         and there is an explicit label given for missings,
         output an observation to handle the missing case;
      %IF %LENGTH(&MISSING)>0 %then
        %do;
           if not foundmissing then
             do;
                label="&MISSING";
                output;
             end;
        %end;

      * Handle the "Other" cases.  The special variable "HLO"
        must be assigned the letter "O";
      hlo='O';
      label="&OTHER";
      output;
  end;
```

```
 keep &VALUES label FMTNAME hlo;
 rename &VALUES = start;
run;

proc format library=&LIBRARY cntlin=xxxx;
run;
%END;

%MEND;
```

SOURCE CODE FOR %TRANSPO

```
%MACRO TRANSPO(data=,out=,by=,vars=,id=,
               varsep=,modlabel=Y,labelsep=,copyvars=);

   /* last modified 8/4/2006
                                                                  72nd col -->|
 Function: Transpose many observations into one per by-group.
       It handles both character and numeric variables and preserves
       the original variable attributes.

 Author:  Ted Clay
          Clay Software and Statistics, Ashland, OR
          541-482-6435
          tclay@ashlandhome.net
          www.ashlandinternet.com/~tedclay
    Please keep, use and pass on the TRANSPO macro with this authorship
        note. (Email the author with any improvements.)

 Reference:
     "A Better Transpose: %TRANSPO Macro Preserves Variable Attributes"
               WUSS11 Proceedings, San Francisco, Nov 5-7, 2003.

 Background: This is different from PROC TRANSPOSE in that
      1) It preserves the original variable attributes, 2) It outputs
      one observation per by-group, and 3) It does not change variables
      into observations.

 Parameters:
         DATA = (required) Name of input dataset.  Must be
                  sorted by the BY variable(s).
         OUT  = (required) Name for output dataset.
         BY   = (required) By-grouping variable(s).  Output dataset
                  will have one obs per unique value.
         VARS = (required) List of one or more variables to be
                  transposed.  Can be any character or numeric variables.
                  The list can be any valid SAS variable list,
                  e.g. x1-x5, age--sex, etc.

         (optional parameters:)

         ID   = Name of variable whose values will determine
                the suffix of the new variable names.  May be character
                or numeric.  If it has a format assigned, the formatted
                values are used.  If no ID variable is given, the
                suffixes of the new variables will be the number 1,2,3,
                etc. assigned to the observations in sequence.
                If ID is specified, the input dataset should have at
                most one (formatted) value of the ID variable per
                By-group.
         MODLABEL = Y/N. (Deflault=Y). If Y, the new labels are the old
                plus the (formatted) value of the ID variable.
```

```
        VARSEP   = Text to insert between the old variable name and
                   the variable suffix being added.
        LABELSEP = Text to insert between the old label and
                   the variable suffix being added.
                   Only used if MODLABEL=Y.
        COPYVARS = List of vars to be copied from the last obs in
               the by-group, with no name change.  This is appropriate
               for handling additional variables that are the same
               on all observations within the by-group, but you
               prefer not to list them on the BY parameter.

Additional Feature:
      For each variable on the VAR parameter, a global macro variable
      of the same name is created containing the list of output
      variables derived from it.  These macro variables can be used in
      array statements in a data step processing the output data set.
      If the original variable XXX was transposed into variables
      XXX_A XXX_B and XXX_C, the global macro variable XXX would be
      assigned the value:  XXX_A XXX_B XXX_C;
      In a subsequent step the array statement could be written as:
              array XXX_ &XXX;

Details about the ID variable:
    Case 1: No ID variable specified.  The macro attaches the
            suffixes "1","2","3", etc. to the transposed variables.
    Case 2: ID variable specified.  The macro uses the values of the
               variable to assign the suffix of the transposed data.
               If the ID variable has a format assigned to it, the
               formatted values of the ID variable are used.  Invalid
               characters in the formatted or unformatted ID variable
               are replaced with underscores for the new variable
               names, but are not replaced with underscores when
               appended to the existing variable labels.

An error results if the ID variable has blank formatted values or
      is not formatted and has missing values.

Restrictions:
    1. The OBS= or FIRSTOBS= options may be in effect, because they
          conflict with the WHERE= option used in the macro.
    2. The input data set may not have data set options specified.
    3. Handles 4 digits of id values (that is, up to 9999) per by-group.
       Presumably long before that limit is reached, SAS will reach
       its limit on the number of datasets that can be on a MERGE
       statement, or will run out of memory.
    4. Handles up to 4 digits (9999) variables to be transposed.
    5. The number of variables on the output dataset will be ...
       <number of ID values> times <number of variables in VARS
       parameter> plus <number of BY variables> plus
       <number of COPYVAR variables>.  SAS will impose a limit on the
       maximum number of variables on a data set, varying depending on
       the SAS version.
    6. Variable name literals are not supported.

User responsibilities:
    1. When you concatenate the longest old variable name, plus the
       VARSEP, plus the longest (formatted) ID variable value, the
       result should be within the limit imposed by SAS.  This is 32
       characters unless the VALIDVARNAME=V6 option is in effect, in
       which case it is 8 characters.
    2. The VARS, ID and BY parameters each must be a valid SAS variable
       list for the input dataset.  If not, the first SAS error message
       will give a reasonable explanation of any problem of this kind.
```

```
  */

*-------------------------------------------------------------------;
*                   CHECKING PARAMETERS
*-------------------------------------------------------------------;

%LET PARMERR=NO;

%* Check that BY variable(s) are specified;
%IF %STR(&BY) EQ %THEN
  %DO;
      %LET PARMERR=YES;
      %PUT ERROR: The BY parameter is required but missing.;
      %PUT The TRANSPO stopped before creating an output dataset.;
  %END;

%* Check that VARS variable(s) are specified;
%IF %QUOTE(&VARS) EQ %THEN
  %DO;
      %LET PARMERR=YES;
      %PUT ERROR: The VARS parameter is required but missing.;
      %PUT The TRANSPO stopped before creating an output dataset.;
  %END;

%* Note: Although we say the data and out parameters are required,
     no error happens if either are left blank, so we do not check them.
     It is good programming practice to include them explicitly.;

%IF %STR(&VARSEP) NE %THEN
  %DO;
    %LET BADCOL=%SYSFUNC(verify(
            %UPCASE(&VARSEP),ABCDEFGHIJKLMNOPQRSTUVWXYZ_0123456789));
    %IF &BADCOL %THEN
    %DO;
        %LET PARMERR=YES;
        %PUT The VARSEP parameter "&VARSEP" contains an invalid
character at column &BADCOL.;
        %PUT The TRANSPO stopped before creating an output dataset.;
    %END;
  %END;

%* If any problem was found, jump to the bottom of the macro;
%IF &PARMERR=YES %THEN %GOTO DONE;

%* All variables used in parameters must be on the input dataset.
   If not, the following step will cause a SAS error;
proc contents data=&DATA(keep=&BY &VARS &ID) noprint out=zzconts;
run;

%*-------------------------------------------------------------------;
%*   SECTION 1:    PRELIMINARY PART OF MACRO
%*-------------------------------------------------------------------;

%* Get last by-variable;

  %DO V=1 %TO 99;
    %LET WORD = %SCAN(&BY,&V,' -');
    %IF %STR(&WORD) NE %THEN %LET LASTBY=&WORD;
    %ELSE %LET V=100;
  %END;


%*-------------------------------------------------------------------;
%* CASE 1: No ID variable specified.
```

```
%*
%*    Create a view which sequentially numbers the observations within
%*    each by-group, and treat that counter as a numeric ID variable.
%*-------------------------------------------------------------------;

%IF %str(&ID) = %THEN
%DO;

%* Define a view of the data with a variable going 1,2,3... within each
   by-group;

data _zzseq_v / view=_zzseq_v;
 set &DATA;
 by &BY;
 if first.&LASTBY then _zzseq = 0;
 _zzseq +1;
run;

%* Set up macro variables to tell the final step how to operate;

%LET IDTYPE=NUM;
%LET IDFMTED=NO;
%LET READFROM=_zzseq_v;
%LET IDVAR=_zzseq;

%END;

%ELSE %DO;

%*-------------------------------------------------------------------;
%* CASE 2: An ID variable was specified.
%*-------------------------------------------------------------------;

* Check the type of the ID variable, and store its format and
   format length.;

data _null_;
 set zzconts;
 if upcase(name)=upcase("&ID") then
   do;
     if      type=1 then call symput('IDTYPE','NUM');
     else if type=2 then call symput('IDTYPE','CHAR');

     if format=' ' then call symput('IDFMTED','NO');
     else               call symput('IDFMTED','YES');

     call symput('IDFMT' ,trim(left(format )));
     call symput('IDFMTL',trim(left(formatl)));

   end;
run;

%put idtype is &idtype;
%put idfmted is &idfmted;
%put idfmt is &idfmt;
%put idfmtl is &idfmtl;

%IF %STR(&IDFMTL)=0 %THEN %LET IDFMTL=;

%LET READFROM=&DATA;
%LET IDVAR=&ID;

%END;
```

```
%*--------------------------------------------------------------------;
%*  SECTION 2:       MAIN PART OF MACRO
%*--------------------------------------------------------------------;

%*--------------------------------------------------------------------;
%* 1.  Use Proc Freq to get the list of values of ID variable.
%*--------------------------------------------------------------------;
proc freq data=&READFROM;
 table &IDVAR / noprint out=idfreqs;
run;

%LET BLANKID=NO;

* Store the ID variable values (formatted) into a series of macro vars;
data idfreqs2;
 set idfreqs;
 length compare $200 CLEAN $200;

 * Calculate values to compare data with.
   If ID variable is formatted, must compare with formatted values,
   because many unformatted values may share the same formatted value.
   No BY-group should have more than one observation with the same
   formatted value of the ID variable.;

 %IF &IDFMTED=YES %THEN
                      compare=left(put(&idvar,&IDFMT&IDFMTL..))%str(;);
 %ELSE %IF &IDTYPE=NUM  %THEN compare = left(put(&IDVAR,best.))%str(;);
 %ELSE %IF &IDTYPE=CHAR %THEN compare = left(&IDVAR)%str(;);

 * If ID is formatted, we look for blank in the COMPARE variable.  The
    missing value of the ID might get formatted to a value label such
    as "Miss".  That would be OK.  But if there is no format,
    we reject if there is a missing value in the ID variable.  ;
 if
   %IF &IDFMTED=YES %THEN missing(compare);
   %ELSE                  missing(&IDVAR);
    then call symput('BLANKID','YES');

 * Calculate "cleaned up" id value valid to use in variable names.
   Invalid characters are replaced with the underscore.;
 clean = compare;
 do i = 1 to length(trim(clean));
   if verify(substr(upcase(clean),i,1),
                     'ABCDEFGHIJKLMNOPQRSTUVWXYZ_0123456789')>0
     then substr(clean,i,1)='_';
 end;

 * Store the "Compare" and "Cleaned" values as macro variables;
 length macname $ 8;
 macname='ID'||left(put(_n_,4.));
 call symput(macname,trim(left(compare)));

 macname='CLN'||left(put(_n_,4.));
 call symput(macname,trim(left(clean)));

 * Store number of (formatted) values of ID variable;
 call symput('NUMIDS',left(put(_n_,4.)));

run;

%IF %QUOTE(&BLANKID)=YES %THEN
 %DO;
     %PUT ERROR: ID variable &ID has a missing value or blank
 formatted value.;
```

```
     %PUT ERROR:   TRANSPO macro will stop executing.;
     %GOTO DONE;
 %END;

%* "Super-quote" the raw ID values because they may contain ampersand
    or percent signs.  Otherwise the macro processor would attempt to
    resolve these in the WHERE= clause of the final data step;
%DO I=1 %TO &NUMIDS;
  %LET ID&I=%SUPERQ(ID&I);
%END;

%*------------------------------------------------------------------------;
%* 2. Store the names and labels of the variables to be transposed
%*           into a set of macro variables;
%*------------------------------------------------------------------------;

%* This allows VARS to be any VALID form of a SAS variable list;

proc contents data=&READFROM(keep=&VARS) noprint out=xxconts;
run;

data _null_;
 set xxconts;
 * Because of the next statement, the output variables will all have
   labels, even if input variables did not.
   The default label is the variable name. ;
 if label=' ' then label=name;

 * Store variable names and labels into macro arrays;
 length macname $ 8;
 macname='VAR'||left(put(_n_,4.));
 call symput(macname  ,trim(left(name)));
 macname='LBL'||left(put(_n_,4.));
 call symput(macname  ,trim(left(label)));
 call symput('NUMVARS',left(put(_n_,4.)));
run;

%* "Super-quote" the variable labels because they may contain ampersand
    or percent signs.  Otherwise the macro processor would
    attempt to resolve these in the final data step;
%DO V=1 %TO &NUMVARS;
  %LET LBL&V=%SUPERQ(LBL&V);
%END;

%*------------------------------------------------------------------------;
%* 3.  Create the output dataset by merging the input many
%*             times with itself.
%*------------------------------------------------------------------------;

%* Note: For each different value of the ID variable, we merge in a copy
       of the input data using a WHERE option which selects only the
       records where the ID variable has that value.  The form of the
       WHERE option must vary depending on the characteristics of the
       ID variable.  The variables to be transposed are renamed using
       the rename= dataset option, adding a suffix which is the same as
       the value used in the WHERE option. ;

data &OUT;
  merge
    %DO I=1 %TO &NUMIDS;
      &READFROM(keep=&BY &IDVAR &VARS &COPYVARS
      %LET IDVAL=%NRBQUOTE(&&ID&I);
      %IF       &IDFMTED=YES %THEN
      where=(left(put(&IDVAR,&IDFMT&IDFMTL..)) = "&IDVAL");
```

```
      %ELSE %IF &IDTYPE=NUM  %THEN
                 where=(     &IDVAR  =  &&ID&I );
      %ELSE %IF &IDTYPE=CHAR %THEN
                 where=(left(&IDVAR) = "%NRQUOTE(&&ID&I)");

                 rename=(
                  %DO V=1 %TO &NUMVARS;
                       &&VAR&V = &&VAR&V..&VARSEP&&CLN&I
                  %END;
                 )  )
    %END;
     ;     %* end of merge statement;
  by &BY;

  if not (first.&LASTBY and last.&LASTBY) then
    do;
        put 'ERROR: More than one observation has the same formatted'
         /  '   value of the ID variable within the same by-group';
        _error_=1;
        stop;
    end;

  %* Put a blank at the end of LABELSEP if it is present;
  %IF %STR(&LABELSEP) NE %THEN %LET LABELSEP=%STR(&LABELSEP );

  %IF &MODLABEL=Y %THEN
     %DO I=1 %TO &NUMIDS;
         %DO V=1 %TO &NUMVARS;
             label &&VAR&V..&Varsep&&CLN&I = "&&LBL&V &LABELSEP&&ID&I";
         %END;
     %END;
  drop &IDVAR;
run;

%*------------------------------------------------------------------------;
%* 4.  Store transposed variable names in global macro variables
%*     for use in subsequent program code, such as array statements.
%*        See "Additional Feature" in header documentation.
%*------------------------------------------------------------------------;

%LOCAL STEM;
%GLOBAL &VARS;
 %DO V=1 %TO &NUMVARS;
     %LET STEM=&&VAR&V;
     %LET &STEM=;
     %DO I=1 %TO &NUMIDS;
          %LET &STEM = &&&STEM &&VAR&V..&Varsep&&CLN&I;
     %END;
 %END;

%DONE: ;

%MEND;
```

SOURCE CODE FOR %ARRAY (ALSO USES %NUMLIST)

```
%MACRO ARRAY(arraypos, array=, data=, var=, values=,
                      delim=%STR( ), debug=N, numlist=Y);

 /* last modified 8/4/2006                    a.k.a. MACARRAY( ).
                                                        72nd col -->|
 Function: Define one or more Macro Arrays
     This macro creates one or more macro arrays, and stores in them
     character values from a SAS dataset or view, or an explicit list
     of values.

     A macro array is a list of macro variables sharing the same prefix
     and a numerical suffix.  The suffix numbers run from 1 up to a
     highest number.  The value of this highest number, or the length
     of the array, is stored in an additional macro variable with the
     same prefix, plus the letter “N”.  The prefix is also referred to
     as the name of the macro array. For example, "AA1", "AA2", "AA3",
     etc., plus "AAN".  All such variables are declared GLOBAL.

 Authors: Ted Clay, M.S.   tclay@ashlandhome.net  (541) 482-6435
          David Katz, M.S. www.davidkatzconsulting.com
      "Please keep, use and pass on the ARRAY and DO_OVER macros with
          this authorship note.  -Thanks "

 Full documentation with examples appears in SUGI Proceedings, 2006,
     "Tight Looping With Macro Arrays" by Ted Clay
 Please send improvements, fixes or comments to Ted Clay.

 Parameters:
    ARRAYPOS and
    ARRAY are equivalent parameters.  One or the other, but not both,
              is required.  ARRAYPOS is the only position parameter.
            = Identifier(s) for the macro array(s) to be defined.
    DATA = Dataset containing values to load into the array(s).  Can be
              a view, and dataset options such as WHERE= are OK.
    VAR  = Variable(s) containing values to put in list. If multiple
              array names are specified in ARRAYPOS or ARRAY then the
              same number of variables must be listed.
    VALUES  = An explicit list of character strings to put in the list
              or lists.  If present, VALUES are used rather than DATA
              and VAR.  VALUES can be a numbered list, eg 1-10, a01-A20,
              a feature which can be turned of with NUMLIST=N.
              The VALUES can be used with one or more array names
              specified in the ARRAYPOS or ARRAY parameters.  If more
              than one array name is given, the values are assigned to
              each array in turn.  For example, if arrays AA and BB
              are being assigned values, the values are assigned to
              AA1, BB1, AA2, BB2, AA3, BB3, etc.  Therefore the number
              of values must be a multiple of the number of arrays.

    DELIM = Character used to separate values in VALUES parameter.
              Blank is default.

    DEBUG = N/Y. Default=N.  If Y, debugging statements are activated.

    NUMLIST = Y/N.  Default=Y.  If Y, VALUES may be a number list.

 REQUIRED OTHER MACRO: Requires NUMLIST if using numbered lists are used
              in the VALUES parameter.

 How the program works.
```

```
    When the VALUES parameter is used, it is parsed into individual
    words using the scan function. With the DATA parameter, each
    observation of data to be loaded into one or more macro
    arrays, _n_ determines the numeric suffix.  Each one is declared
    GLOBAL using "call execute" which is acted upon by the SAS macro
    processor immediately. (Without this "global" setting, "Call symput"
    would by default put the new macro variables in the local symbol
    table, which would not be accessible outside this macro.)  Because
    "call execute" only is handling macro statements, the following
    statement will normally appear on the SAS log: "NOTE: CALL EXECUTE
    routine executed successfully, but no SAS statements were generated."

 History
  7/14/05 handle char variable value containing single quote
  1/19/06 VALUES can be a a numbered list with dash, e.g. AA1-AA20
  4/1/06 simplified process of making variables global.
  4/12/06 allow VALUES= when creating more than one macro array.

    */

%LOCAL prefixes PREFIXN manum _VAR_N iter i J val VAR WHICH MINLENG
   PREFIX1 PREFIX2 PREFIX3 PREFIX4 PREFIX5 PREFIX6 PREFIX7 PREFIX8
   PREFIX9 PREFIX10 PREFIX11
   var1 var2 var3 var4 var5 var6 var7 var8 var9 var10 var11 ;

%* Get array names from either the keyword or positional parameter;
%if &ARRAY= %then %let PREFIXES=&ARRAYPOS;
%else %let PREFIXES=&ARRAY;

%* Parse the list of macro array names;
%do MANUM = 1 %to 999;
 %let prefix&MANUM=%scan(&prefixes,&MAnum,' ');
 %if &&prefix&MANUM ne %then
   %DO;
    %let PREFIXN=&MAnum;
    %global &&prefix&MANUM..N;
    %* initialize length to zero;
    %let &&prefix&MANUM..N=0;
   %END;
  %else %goto out1;
%end;
%out1:
%if &DEBUG=Y %then %put PREFIXN is &PREFIXN;
%* Parse the VAR parameter;
%let _VAR_N=0;
%do MANUM = 1 %to 999;
 %let _var_&MANUM=%scan(&VAR,&MAnum,' ');
 %if %str(&&_var_&MANUM) ne %then %let _VAR_N=&MAnum;
 %else %goto out2;
%end;
%out2:
%IF &PREFIXN=0 %THEN
    %PUT ERROR: No macro array names are given;
%ELSE %IF %LENGTH(%STR(&DATA)) >0 and &_VAR_N=0 %THEN
    %PUT ERROR: DATA parameter is used but VAR parameter is blank;
%ELSE %IF %LENGTH(%STR(&DATA)) >0 and &_VAR_N ne &PREFIXN %THEN
    %PUT ERROR: The number of variables in the VAR parameter is not
 equal to the number of arrays;
%ELSE %DO;

%*-------------------------------------------------------;
%*  CASE 1: VALUES parameter is used
%*-------------------------------------------------------;
```

```
%IF %LENGTH(%STR(&VALUES)) >0 %THEN
%DO;
     %IF &NUMLIST=Y %then
     %DO;
         %* Check for numbered list of form xxx-xxx and expand it using
             the NUMLIST macro.;
         %IF (%INDEX(%quote(&VALUES),-) GT 0) and
             (%length(%SCAN(%quote(&VALUES),1,-))>0) and
             (%length(%SCAN(%quote(&VALUES),2,-))>0) and
             (%length(%SCAN(%quote(&VALUES),3,-))=0)
           %THEN %LET VALUES=%NUMLIST(&VALUES);
     %END;

%LET MINLENG=99999;
%DO J=1 %TO &PREFIXN;
%DO ITER=1 %TO 9999;
  %LET WHICH=%EVAL((&ITER-1)*&PREFIXN +&J);
  %LET VAL=%SCAN(%STR(&VALUES),&WHICH,%STR(&DELIM));
  %IF %QUOTE(&VAL) NE %THEN
    %DO;
      %GLOBAL &&&&PREFIX&J..&ITER;
      %LET &&&&PREFIX&J..&ITER=&VAL;
      %LET &&&&PREFIX&J..N=&ITER;
    %END;
  %ELSE %goto out3;
%END;
%out3: %IF &&&&&&PREFIX&J..N LT &MINLENG
          %THEN %LET MINLENG=&&&&&&PREFIX&J..N;
%END;

%if &PREFIXN >1 %THEN
%DO J=1 %TO &PREFIXN;
    %IF &&&&&&PREFIX&J..N NE &MINLENG %THEN
%PUT ERROR: Number of values must be a multiple of the number of arrays;
%END;

%END;
%ELSE %DO;

%*-------------------------------------------------------;
%*  CASE 2: DATA and VAR parameters used
%*-------------------------------------------------------;

%* Get values from one or more variables in a dataset or view;
  data _null_;
  set &DATA end = lastobs;
%DO J=1 %to &PREFIXN;
  call execute('%GLOBAL '||"&&PREFIX&J.."||left(put(_n_,5.)) );
  call symput(compress("&&prefix&J"||left(put(_n_,5.))),
              trim(left(&&_VAR_&J)));
  if lastobs then
   call symput(compress("&&prefix&J"||"N"), trim(left(put(_n_,5.))));
%END;
  run ;

%* Write message to the log;
%IF &DEBUG=Y %then
%DO J=1 %to &PREFIXN;
 %PUT &&&&PREFIX&J..N is &&&&&&PREFIX&J..N;
%END;
%END;
%END;
%MEND;
```

SOURCE CODE FOR %DO_OVER (ALSO USES %NUMLIST)

```
%MACRO DO_OVER(arraypos, array=,
               values=, delim=%STR( ),
               phrase=?, escape=?, between=,
               macro=, keyword=);

 /*  Last modified: 8/4/2006
                                                                  72nd col -->|
  Function: Loop over one or more arrays of macro variables
            substituting values into a phrase or macro.

  Authors: Ted Clay, M.S.
              Clay Software & Statistics
              tclay@ashlandhome.net  (541) 482-6435
            David Katz, M.S. www.davidkatzconsulting.com
          "Please keep, use and pass on the ARRAY and DO_OVER macros with
               this authorship note.  -Thanks "
           Send any improvements, fixes or comments to Ted Clay.

  Full documentation with examples appears in
     "Tight Looping with Macro Arrays".SUGI Proceedings 2006,
       The keyword parameter was added after the SUGI article was written.

  REQUIRED OTHER MACROS:
        NUMLIST -- if using numbered lists in VALUES parameter.
        ARRAY   -- if using macro arrays.

  Parameters:

     ARRAYPOS and
     ARRAY are equivalent parameters.  One or the other, but not both,
             is required.  ARRAYPOS is the only position parameter.
           = Identifier(s) for the macro array(s) to iterate over.
             Up to 9 array names are allowed. If multiple macro arrays
             are given, they must have the same length, that is,
             contain the same number of macro variables.

     VALUES = An explicit list of character strings to put in an
              internal macro array, VALUES may be a numbered lists of
              the form 3-15, 03-15, xx3-xx15, etc.

     DELIM = Character used to separate values in VALUES parameter.
             Blank is default.

     PHRASE = SAS code into which to substitute the values of the
              macro variable array, replacing the ESCAPE
              character with each value in turn.  The default
              value of PHRASE is a single <?> which is equivalent to
              simply the values of the macro variable array.
              The PHRASE parameter may contain semicolons and extend to
              multiple lines.
              NOTE: The text "?_I_", where ? is the ESCAPE character,
                    will be replaced with the value of the index variable
                    values, e.g. 1, 2, 3, etc.
              Note: Any portion of the PHRASE parameter enclosed in
                single quotes will not be scanned for the ESCAPE.
                So, use double quotes within the PHRASE parameter.

              If more than one array name is given in the ARRAY= or
              ARRAYPOS parameter, in the PHRASE parameter the ESCAPE
              character must be immediately followed by the name of one
              of the macro arrays, using the same case.
```

```
   ESCAPE = A single character to be replaced by macro array values.
            Default is "?".

   BETWEEN = code to generate between iterations of the main
            phrase or macro.  The most frequent need for this is to
            place a comma between elements of an array, so the special
            argument COMMA is provided for programming convenience.
            BETWEEN=COMMA is equivalent to BETWEEN=%STR(,).

   MACRO = Name of an externally-defined macro to execute on each
            value of the array. It overrides the PHRASE parameter.
            The parameters of this macro may be a combination of
            positional or keyword parameters, but keyword parameters
            on the external macro require the use of the KEYWORD=
            parameter in DO_OVER.  Normally, the macro would have
            only positional parameters and these would be defined in
            in the same order and meaning as the macro arrays specified
            in the ARRAY or ARRAYPOS parameter.
            For example, to execute the macro DOIT with one positional
            parameter, separately define
                     %MACRO DOIT(STRING1);
                         <statements>
                     %MEND;
            and give the parameter MACRO=DOIT.  The values of AAA1,
            AAA2, etc. would be substituted for STRING.
            MACRO=DOIT is equivalent to PHRASE=%NRQUOTE(%DOIT(?)).
            Note: Within an externally defined macro, the value of the
            macro index variable would be coded as "&I".  This is
            comparable to "?_I_" within the PHRASE parameter.

  KEYWORD = Name(s) of keyword parameters used in the definition of
            the macro refered to in the MACRO= parameter. Optional.
            This parameter controls how DO_OVER passes macro array
            values to specific keyword parameters on the macro.
            This allows DO_OVER to execute a legacy or standard macro.
            The number of keywords listed in the KEYWORD= parameter
            must be less than or equal to the number of macro arrays
            listed in the ARRAYPOS or ARRAY parameter.  Macro array
            names are matched with keywords proceeding from right
            to left.  If there are fewer keywords than macro array
            names, the remaining array names are passed as positional
            parameters to the external macro.  See Example 6.

Rules:
    Exactly one of ARRAYPOS or ARRAY or VALUES is required.
    PHRASE or MACRO is required.  MACRO overrides PHRASE.
    ESCAPE is used when PHRASE is used, but is ignored with MACRO.
    If ARRAY or ARRAYPOS have multiple array names, these must exist
        and have the same length.  If used with externally defined
        MACRO, the macro must have positional parameters that
        correspond 1-for-1 with the array names.  Alternatively, one
        can specify keywords which tell DO_OVER the names of keyword
        parameters of the external macro.

Examples:
   Assume macro array AAA has been created with
           %ARRAY(AAA,VALUES=x y z)
    (1) %DO_OVER(AAA) generates: x y z;
    (2) %DO_OVER(AAA,phrase="?",between=comma) generates: "x","y","z"
    (3) %DO_OVER(AAA,phrase=if L="?" then ?=1;,between=else) generates:
                  if L="x" then x=1;
             else if L="y" then y=1;
             else if L="z" then z=1;
```

```
      (4) %DO_OVER(AAA,macro=DOIT) generates:
                %DOIT(x)
                %DOIT(y)
                %DOIT(z)
            which assumes %DOIT has a single positional parameter.
            It is equivalent to:
            %DO_OVER(AAA,PHRASE=%NRSTR(%DOIT(?)))

      (5) %DO_OVER(AAA,phrase=?pct=?/tot*100; format ?pct 4.1;)
            generates:
                xpct=x/tot*100; format xpct 4.1;
                ypct=y/tot*100; format ypct 4.1;
                zpct=z/tot*100; format zpct 4.1;
      (6) %DO_OVER(aa bb cc,MACRO=doit,KEYWORD=borders columns)
          is equivalent to %DO_OVER(aa,bb,cc,
                   PHRASE=%NRSTR(%doit(?aa,borders=?bb,columns=?cc)))
          Either example would generate the following internal do-loop:
          %DO I=1 %to &AAN;
            %doit(&&aa&I,borders=&&bb&I,columns=&&cc&I)
          %END;
          Because we are giving three macro array names, the macro DOIT
          must have three parameters.  Since there are only two keyword
          parameters listed, the third parameter is assumed to be
          positional.  Positional parameters always preceed keyword
          parameters in SAS macro definitions, so the first parameter
          a positional parameter, which is given the values of first
          macro array "aa".  The second is keyword parameter "borders="
          which is fed the values of the second array "bb".  The third
          is a keyword parameter "columns=" which is fed the values of
          the third array "cc".

  History
    7/15/05 changed %str(&VAL) to %quote(&VAL).
    4/1/06 added KEYWORD parameter
    4/9/06 declared "_Intrnl" array variables local to remove problems
           with nesting with VALUES=.
    8/4/06 made lines 72 characters or less to be mainframe compatible
*/

%LOCAL
  _IntrnlN
  _Intrnl1  _Intrnl2  _Intrnl3  _Intrnl4  _Intrnl5
  _Intrnl6  _Intrnl7  _Intrnl8  _Intrnl9  _Intrnl10
  _Intrnl11 _Intrnl12 _Intrnl13 _Intrnl14 _Intrnl15
  _Intrnl16 _Intrnl17 _Intrnl18 _Intrnl19 _Intrnl20
  _Intrnl21 _Intrnl22 _Intrnl23 _Intrnl24 _Intrnl25
  _Intrnl26 _Intrnl27 _Intrnl28 _Intrnl29 _Intrnl30
  _Intrnl31 _Intrnl32 _Intrnl33 _Intrnl34 _Intrnl35
  _Intrnl36 _Intrnl37 _Intrnl38 _Intrnl39 _Intrnl40
  _Intrnl41 _Intrnl42 _Intrnl43 _Intrnl44 _Intrnl45
  _Intrnl46 _Intrnl47 _Intrnl48 _Intrnl49 _Intrnl50
  _Intrnl51 _Intrnl52 _Intrnl53 _Intrnl54 _Intrnl55
  _Intrnl56 _Intrnl57 _Intrnl58 _Intrnl59 _Intrnl60
  _Intrnl61 _Intrnl62 _Intrnl63 _Intrnl64 _Intrnl65
  _Intrnl66 _Intrnl67 _Intrnl68 _Intrnl69 _Intrnl70
  _Intrnl71 _Intrnl72 _Intrnl73 _Intrnl74 _Intrnl75
  _Intrnl76 _Intrnl77 _Intrnl78 _Intrnl79 _Intrnl80
  _Intrnl81 _Intrnl82 _Intrnl83 _Intrnl84 _Intrnl85
  _Intrnl86 _Intrnl87 _Intrnl88 _Intrnl89 _Intrnl90
  _Intrnl91 _Intrnl92 _Intrnl93 _Intrnl94 _Intrnl95
  _Intrnl96 _Intrnl97 _Intrnl98 _Intrnl99 _Intrnl100
 _KEYWRDN _KEYWRD1 _KEYWRD2 _KEYWRD3 _KEYWRD4 _KEYWRD5
 _KEYWRD6 _KEYWRD7 _KEYWRD8 _KEYWRD9
 _KWRDI
```

```
 ARRAYNOTFOUND CRC CURRPREFIX DELIMI DID FRC I ITER J KWRDINDEX MANUM
 PREFIXES PREFIXN PREFIX1 PREFIX2 PREFIX3 PREFIX4 PREFIX5
 PREFIX6 PREFIX7 PREFIX8 PREFIX9
 SOMETHINGTODO TP VAL VALUESGIVEN
 ;

%let somethingtodo=Y;

%* Get macro array name(s) from either keyword or positional parameter;
%if       %str(&arraypos) ne %then %let prefixes=&arraypos;
%else %if %str(&array)    ne %then %let prefixes=&array;
%else %if %quote(&values) ne %then %let prefixes=_Intrnl;
%else %let Somethingtodo=N;

%if &somethingtodo=Y %then
%do;

%* Parse the macro array names;
%let PREFIXN=0;
%do MAnum = 1 %to 999;
 %let prefix&MANUM=%scan(&prefixes,&MAnum,' ');
 %if &&prefix&MAnum ne %then %let PREFIXN=&MAnum;
 %else %goto out1;
%end;
%out1:

%* Parse the keywords;
%let _KEYWRDN=0;
%do _KWRDI = 1 %to 999;
 %let _KEYWRD&_KWRDI=%scan(&KEYWORD,&_KWRDI,' ');
 %if &&_KEYWRD&_KWRDI ne %then %let _KEYWRDN=&_KWRDI;
 %else %goto out2;
%end;
%out2:

%* Load the VALUES into macro array 1 (only one is permitted);
%if %length(%str(&VALUES)) >0 %then %let VALUESGIVEN=1;
%else %let VALUESGIVEN=0;
%if &VALUESGIVEN=1 %THEN
%do;
         %* Check for numbered list of form xxx-xxx and expand it
            using NUMLIST macro.;
         %IF (%INDEX(%STR(&VALUES),-) GT 0) and
             (%SCAN(%str(&VALUES),2,-) NE ) and
             (%SCAN(%str(&VALUES),3,-) EQ )
           %THEN %LET VALUES=%NUMLIST(&VALUES);

%do iter=1 %TO 9999;
  %let val=%scan(%str(&VALUES),&iter,%str(&DELIM));
  %if %quote(&VAL) ne %then
    %do;
      %let &PREFIX1&ITER=&VAL;
      %let &PREFIX1.N=&ITER;
    %end;
  %else %goto out3;
%end;
%out3:
%end;

%let ArrayNotFound=0;
%do j=1 %to &PREFIXN;
  %*put prefix &j is &&prefix&j;
  %LET did=%sysfunc(open(sashelp.vmacro
                   (where=(name eq "%upcase(&&PREFIX&J..N)")) ));
```

```
  %LET frc=%sysfunc(fetchobs(&did,1));
  %LET crc=%sysfunc(close(&did));
  %IF &FRC ne 0 %then
    %do;
       %PUT Macro Array with Prefix &&PREFIX&J does not exist;
       %let ArrayNotFound=1;
    %end;
%end;

%if &ArrayNotFound=0 %then %do;

%if %quote(%upcase(&BETWEEN))=COMMA %then %let BETWEEN=%str(,);

%if %length(%str(&MACRO)) ne 0 %then
  %do;
     %let TP = %nrstr(%&MACRO)(;
     %do J=1 %to &PREFIXN;
         %let currprefix=&&prefix&J;
         %IF &J>1 %then %let TP=&TP%str(,);
            %* Write out macro keywords followed by equals.
               If fewer keywords than macro arrays, assume parameter
               is positional and do not write keyword=;
            %let kwrdindex=%eval(&_KEYWRDN-&PREFIXN+&J);
            %IF &KWRDINDEX>0 %then %let TP=&TP&&_KEYWRD&KWRDINDEX=;
         %LET TP=&TP%nrstr(&&)&currprefix%nrstr(&I);
     %END;
     %let TP=&TP);  %* close parenthesis on external macro call;
  %end;
%else
  %do;
     %let TP=&PHRASE;
     %let TP = %qsysfunc(tranwrd(&TP,&ESCAPE._I_,%nrstr(&I.)));
     %let TP = %qsysfunc(tranwrd(&TP,&ESCAPE._i_,%nrstr(&I.)));
     %do J=1 %to &PREFIXN;
         %let currprefix=&&prefix&J;
         %LET TP = %qsysfunc(tranwrd(&TP,&ESCAPE&currprefix,
                                %nrstr(&&)&currprefix%nrstr(&I..)));
         %if &PREFIXN=1 %then %let TP = %qsysfunc(tranwrd(&TP,&ESCAPE,
                                %nrstr(&&)&currprefix%nrstr(&I..)));
     %end;
  %end;

%* resolve TP (the translated phrase) and perform the looping;
%do I=1 %to &&&prefix1.n;
%if &I>1 and %length(%str(&between))>0 %then &BETWEEN;
%unquote(&TP)
%end;

%end;
%end;

%MEND;
```

SOURCE CODE FOR %NUMLIST (USED BY %ARRAY AND %DO_OVER)

```
%MACRO NUMLIST(listwithdash);
  /*
                                                              72nd col -->|
    Function: Generate the elements of a numbered list.

              For example, AA1-AA3 generates AA1 AA2 AA3
              No prefix is necessary -- 1-3 generates 1 2 3.

      Author: Ted Clay, M.S.
            Clay Software & Statistics
            tclay@ashlandhome.net  (541) 482-6435
      "Please keep, use and share this macro with this authorship note."

   Parameter:
       ListWithDash -- text string containing a dash.
           The text before the dash, and the text after the dash,
           usually begin with a the same character string, called the
           stem.  (The stem could be blank or null, as is the case of
           number-dash-number.) After the common stem must be two
           numbers.  The first number must be less than the second
           number.  Leading zeroes on the numbers are preserved.

  How it works: The listwithdash is parsed into _before and _after.
         _before and _after are compared equal up to the length of the
         "stem".  What is after the "stem" is assigned to _From and _to,
         which must convert to numerics. Finally, the macro generates
         stem followed by all the numbers from _from through _to

  Examples:
     %numlist(3-6) generates 3 4 5 6.
     %numlist(1993-2004) generates 1993 1994 1995 1996 1997 1998 1999
                                   2000 2001 2002 2003 2004.
     %numlist(var8-var12) generates var8 var9 var10 var11 var12.
     %numlist(var08-var12) generates var08 var09 var10 var11 var12.

  */

%local _before _after _length1 _length2 minlength samepos _pos
       _from _to i;

  %let _before = %scan(%quote(&listwithdash),1,-);
  %let _after  = %scan(%quote(&listwithdash),2,-);
  %let _length1 = %length(%quote(&_before));
  %let _length2 = %length(%quote(&_after));
  %let minlength=&_length1;
  %if &_length2 < &minlength %then %let minlength=&_length2;

%*put before is &_before;
%*put after is &_after;
%*put minlength is &minlength;
  %* Stemlength should be just before the first number or the first
      unequal character;
  %let stemlength=0;
  %let foundit=0;
  %do _pos = 1 %to &minlength;
    %LET CHAR1=%upcase(%substr(%quote(&_before),&_pos,1));
    %LET CHAR2=%upcase(%substr(%quote(&_after ),&_pos,1));
    %if %index(1234567890,%QUOTE(&CHAR1)) GE 1 %then %let ISANUMBER=Y;
    %else %let isanumber=N;
```

```
    %if &foundit=0 and
         ( &isanumber=Y OR %quote(&CHAR1) NE %QUOTE(&CHAR2) )
         %then %do;
             %let stemlength=%EVAL(&_pos-1);
             %*put   after assignment stemlength is &stemlength;
             %let foundit=1;
          %end;
  %end;

  %if &stemlength=0 %then %let stem=;
  %else %let stem = %substr(&_before,1,&stemlength);

  %let _from=%substr(&_before,%eval(&stemlength+1));
  %let _to  =%substr(&_after, %eval(&stemlength+1));

%IF %verify(&_FROM,1234567890)>0 or
    %verify(&_TO  ,1234567890)>0 %then
  %PUT ERROR in NUMLIST macro: Alphabetic prefixes are different;

%else %if &_from <= &_to %then
%do _III_=&_from %to &_to;
    %LET _XXX_=&_iii_;
    %do _JJJ_=%length(&_iii_) %to %eval(%length(&_from)-1);
        %let _XXX_=0&_XXX_;
    %end;
%TRIM(&stem&_XXX_)
%end;
%else %PUT ERROR in NUMLIST macro: From "&_from" not <= To "&_to";

%MEND;
```