Paper BB-07-2013

A Better Way to Flip (Transpose) a SAS® Dataset

Arthur S. Tabachneck, Ph.D., myQNA, Inc., Thornhill, Ontario Canada
Xia Ke Shan, Chinese Financial Electrical Company, Beijing, China
Robert Virgile, Robert Virgile Associates, Inc., Lexington, MA
Joe Whitehurst, High Impact Technologies, Atlanta GA

ABSTRACT

Many SAS® programmers have flipped out when confronted with having to flip (transpose) a SAS dataset, especially if they had to transpose multiple variables, needed transposed variables to be in a specific order, had a mixture of character and numeric variables to transpose, or if they needed to retain a number of non-transposed variables. Wouldn't it be nice to have a way to accomplish such tasks that was easier to understand and modify than PROC TRANSPOSE, was less system resource intensive, required fewer steps and could accomplish the task as much as fifty times or more faster?

THE PROBLEM

If you have ever had to rearrange a SAS dataset, converting it from being a tall dataset to be in the form of a wide dataset, then you are probably already familiar with PROC TRANSPOSE. If you have never confronted such a task, here is a fairly simple example. Suppose you had a dataset like the one produced by the following code and shown in Example 1.

```
data have;
  format idnum 4.;
  input idnum date var1 $;
  informat date date9.;
  format date yymon7.;
  cards;
1 01jan2001 SD
1 01feb2001 EF
1 01mar2001 HK
2 01jan2001 GH
2 01apr2001 MM
2 01may2001 JH
;
```

Example 1

Obs	idnum	date	var1
1	1	2001JAN	SD
2	1	2001FEB	EF
3	1	2001MAR	HK
4	2	2001JAN	GH
5	2	2001APR	MM
6	2	2001MAY	JH

If you had to transform such a file so that it produced the dataset shown in Example 2, the following code is all you would need:

```
PROC TRANSPOSE data=have out=want (drop=_:) prefix=var1_;
  by idnum;
  var var1;
  id date;
run;
```

Example 2

Obs	idnum	var1_2001JAN	var1_2001FEB	var1_2001MAR	var1_2001APR	var1_2001MAY
1	1	SD	EF	HK		
2	2	GH			MM	JH

PROBLEM 1: Speed-Part I

While that seems simple enough, and produces the desired result, we think that a number of SAS users could be losing substantial amounts of time by using PROC TRANSPOSE to accomplish some such tasks. A number of alternative methods have been proposed, including writing datasteps, using other procedures like proc sql and proc summary, and running macros that were designed as PROC TRANSPOSE alternatives, but almost all of the proposals have either had significant limitations, required substantial coding, or necessitated the learning a totally new task. Additionally, few, if any of the proposals have offered a benefit with respect to speed.

While 60 million records probably wouldn't qualify as being representative of *big data* according to today's industry standards, we ran our tests on files of that size as we felt that it would be sufficiently representative of the typical datasets that many of us have to work with. For each comparison, we provide the code, so that you can replicate the comparisons on your own systems using file sizes that are more similar to your typical analytical requirements. However, that said, we believe that the magnitude of the differences we report in this paper will closely approximate the ones that you will find in any tests that you run.

We used the following code to create an expanded Example 1 dataset that was used to compare PROC TRANSPOSE with the macro presented in this paper:

```
data have (drop=i);
  format idnum 32.;
  informat date date9.;
  input date var1 $;
  do i=0 to 9999999;
    if mod(_n_,2) then idnum=2*i+1;
    else idnum=2*i+2;
    output;
  end;
  cards;
01jan2001 SD
01jan2001 GH
01feb2001 EF
01apr2001 MM
01mar2001 HK
01may2001 JH
;

proc sort data=have;
  by idnum date;
run;
```

We compared the performance of two sets of code which, in this case, look almost identical. In fact, the only differences between the two sets of code are that the macro calls for commas instead of semicolons, "=" signs between parameters and values, doesn't need either drop or prefix options, and ends with a right parenthesis:

```
proc transpose data=have out=want (drop=_:) prefix=var1_;
  by idnum;
  var var1;
  id date;
  format date yymon7.;
run;

%transpose(data=have,    out=want,  by=idnum,    var=var1,
           id=date,      format=yymon7., delimiter=_,
           sort=yes,     guessingrows=1000)
```

PROC TRANSPOSE took over nine minutes CPU time to run, while the %transpose macro only took 1 minute. The resulting output files from the two processes were identical. And, since most of our comparisons were accomplished on an older PC that only had 1.5 gb RAM, we replicated some of the tests on a 2 processor server that had 16 gb RAM. The magnitude of the differences was identical and held for files with as few as 300,000 records. Thus, for simple transpositions, PROC TRANSPOSE takes an average of 8½ times longer to run than the %transpose macro.

PROBLEM 2: Inconsistency-Part I

While both approaches had the same result for the Example 1 data, different results would have been obtained if the dates hadn't initially appeared in numeric order. That is, if the dataset had been like the one produced by the following code, and shown in Example 3, the same PROC TRANSPOSE code would produce the result shown in Example 4. The only difference between the two datasets is that in Example 1 the dates across the two *idnums*' records happened to initially appear in numeric order. That is, jan2001 first appeared before feb2001, which appeared before mar2001, etc.

```
data have;
  format idnum 4.;
  input idnum date var1 $;
  informat date date9.;
  format date yymon7.;
  cards;
1 01jan2001 GH
1 01apr2001 MM
1 01may2001 JH
2 01jan2001 SD
2 01feb2001 EF
2 01mar2001 HK
;
```

Example 3

Obs	idnum	date	var1
1	1	2001JAN	GH
2	1	2001APR	MM
3	1	2001MAY	JH
4	2	2001JAN	SD
5	2	2001FEB	EF
6	2	2001MAR	HK

Example 4

Obs	idnum	var1_2001JAN	var1_2001APR	var1_2001MAY	var1_2001FEB	var1_2001MAR
1	1	GH	MM	JH		
2	2	SD			EF	HK

The %transpose macro would have produced the same ordering of variables as shown in Example 2.

PROBLEM 3: Speed-Part II

The output dataset resulting from running PROC TRANSPOSE on Example 3 could be corrected by running an extra datastep that included a retain, length, or format statement before the set statement, thus reordering the variables in ascending date order. However, that would require additional code, thought and time. Specifically, given our example, one would have to write and run code like the following:

```
data want;
  retain idnum
         var1_2001JAN
         var1_2001FEB
         var1_2001MAR
         var1_2001APR
         var1_2001MAY;
  set want;
run;
```

In addition to the time required to write and test the extra code, an additional 26 CPU seconds were needed to run the code for the 60 million record dataset.

PROBLEM 4: System and Programmer Overutilization

If one needs to transpose two or more variables, multiple uses of PROC TRANSPOSE and intermediary datasteps are necessary. Suppose, for example, that you had a dataset like the one produced by the following code and shown in Example 5 on the following page:

```
data have (drop=months);
  format idnum 1.;
  informat date date9.;
  format date date9.;
  input date ind1-ind4 ;
  other=2;
  do idnum=1 to 2;
    date="31dec2010"d;
    do months=3 to 12 by 3;
      date=intnx('month',date,3);
      if not(months eq 9 and mod(idnum,2)) then output;
    end;
  end;
  cards;
01dec2010 1 2 3 4
;
```

Example 5

idnum	date	ind1	ind2	ind3	ind4	other
1	01Mar2011	1	2	3	4	2
1	01Jun2011	1	2	3	4	2
1	01Dec2011	1	2	3	4	2
2	01Mar2011	1	2	3	4	2
2	01Jun2011	1	2	3	4	2
2	01Sep2011	1	2	3	4	2
2	01Dec2011	1	2	3	4	2

Note that while the database structure in Example 5 is far from optimal, it is probably typical of much of the data we all have to work with, especially if we receive tables that were created with spreadsheet programs like Excel. The file contains records for two *IDNUM*s and, for each, there are four *ind*(icator) variables for each quarter. We intentionally designed the file so that the first *IDNUM* was missing data for the month of September. The variable labeled *other* is the same for all months. That is, it is a variable whose replications are irrelevant and the user only wants the variable to be copied rather than transposed. The user wants to create an output dataset like the one shown in Example 6.

Example 6

idnum	other	ind1_Qtr1	ind2_Qtr1	ind3_Qtr1	ind4_Qtr1	ind1_Qtr2	ind2_Qtr2	ind3_Qtr2	ind4_Qtr2	ind1_Qtr3	ind2_Qtr3	ind3_Qtr3	ind4_Qtr3	ind1_Qtr4	ind2_Qtr4	ind3_Qtr4	ind4_Qtr4
1	2	1	2	3	4	1	2	3	4	.	.	.	.	1	2	3	4
2	2	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4

In the case of complex transformations, PROC TRANSPOSE's design requires an inordinate amount of code and resource utilization. The following code would be needed to transform Example 5 into Example 6:

```
proc transpose data=have out=tall ;
  by idnum date;
  var ind1-ind4;
  format date qtr1.;
  copy other;
run;

proc transpose data=tall out=want (where=(not missing(_name_)))
               delimiter=_Qtr;
  by idnum;
  id _name_ date;
  var col1;
  copy other;
run;

data want;
  retain idnum other ind1_Qtr1 ind2_Qtr1 ind3_Qtr1 ind4_Qtr1 ind1_Qtr2
         ind2_Qtr2 ind3_Qtr2 ind4_Qtr2 ind1_Qtr3 ind2_Qtr3 ind3_Qtr3
         ind4_Qtr3 ind1_Qtr4 ind2_Qtr4 ind3_Qtr4 ind4_Qtr4;
  set want (drop=_:);
run;
```

Why were three steps needed? PROC TRANSPOSE was designed to either make a long file wide, or a wide file long. That is, either put all of the variables on one record for each *by* variable, or create one record for every combination of the *by* variable, the *id* variable, any variables to be copied, and only one of the to-be-transposed *var* variable's values.

Example 5 requires both operations in order to accomplish the task with PROC TRANSPOSE. Our example data represents a wide file that we want to make even wider. As such, with PROC TRANSPOSE, we first have to use the procedure to transform the wide file into a longer file, and then use PROC TRANSPOSE, again, to transform the longer file into the ultimately desired width.

Unfortunately, since the first *IDNUM* is missing one month's data, PROC TRANSPOSE will create a file that has the transposed variables in a different order than what the specifications require. Specifically, data for the initially missing month will be output at the end of the record and, as a result, a datastep is needed to rearrange the file to reflect the task's requirements. Even if the dates had all been present, the extra datastep would still have been needed in order to drop the automatic system variable _name_ which had been inserted by PROC TRANSPOSE. The variable couldn't have been dropped in the PROC TRANSPOSE code as it was needed to drop the redundant records that were created as a result of our including the *other* variable in the output requirements.

The code which created the expanded Example 5 dataset that we used to compare the performance of PROC TRANSPOSE and the macro presented in this paper was:

```
data have (drop=months);
  format idnum best32. date date9.;
  informat date date9.;
  input date ind1-ind4 ;
  other=2;
  do idnum=1 to 60000000;
    date="31dec2010"d;
    do months=3 to 12 by 3;
      date=intnx('month',date,3);
      if not(months eq 9 and mod(idnum,2)) then output;
    end;
  end;
  cards;
01dec2010 1 2 3 4
;
```

The above code creates a file that looks like Example 5, but contains data for 60 million *IDNUM*s and takes up 11.5 gb of one's hard drive. The tall file that PROC TRANSPOSE creates takes up an additional 33.3 gb and the output file requires an additional 8.5 gb of space. Additionally, during processing, almost double that amount of disk space is utilized at some points. Thus, in addition to all of the code one has to write, the effort will be meaningless unless the necessary disk space is available.

The code we used to run the macro on both the original and the expanded Example 5 dataset was:

```
%transpose(data=have, out=want, by=idnum, id=date, guessingrows=1000,
           format=qtr1., delimiter=_Qtr, var=ind1-ind4,)
```

As you can see, the code is almost identical to the statements that were needed for PROC TRANSPOSE, but the macro was designed to simply make a wide file wider. The result is that much less typing was necessary and there was no need for an additional 33.3 gb of hard drive space, or any of the additional disk space required by PROC TRANSPOSE. Additionally, unlike PROC TRANPOSE, the output file that resulted from running the macro had the transposed variables in their correct date-based order.

PROBLEM 5: Speed-Part III

The time required to run the three components of the PROC TRANSPOSE solution on the expanded Example 5 dataset surprised us. A total of 110 minutes were needed, while the %transpose macro only took a total of 2 minutes to accomplish the task. We were expecting the macro to perform better, but were not expecting the macro to perform more than fifty times faster. The only unfortunate aspect of testing the macro was that it ran so fast that we didn't even have time to take a coffee break.

PROBLEM 6: Inconsistency-Part II

While a multiple step PROC TRANSPOSE solution can address the transposition of multiple variables, there is a caveat when those variables are of different types. Suppose you had a dataset like the one produced by the following code and shown in Example 7.

```
data have;
  format idnum 4. date date9.;
  input idnum date var1 var2 $;
  informat date date9.;
  cards;
1 31mar2013 1 SD
1 30jun2013 2 EF
1 30sep2013 3 HK
1 31dec2013 4 HL
2 31mar2013 5 GH
2 30jun2013 6 MM
2 30sep2013 7 JH
2 31dec2013 4 MS
;
```

Example 7

idnum	date	var1	var2
1	31MAR2013	1	SD
1	30JUN2013	2	EF
1	30SEP2013	3	HK
1	31DEC2013	4	HL
2	31MAR2013	5	GH
2	30JUN2013	6	MM
2	30SEP2013	7	JH
2	31DEC2013	4	MS

And, let's say that you needed to transpose the data so that it produced the dataset shown in Example 8:

Example 8

idnum	var1_Qtr1	var2_Qtr1	var1_Qtr2	var2_Qtr2	var1_Qtr3	var2_Qtr3	var1_Qtr4	var2_Qtr4
1	1	SD	2	EF	3	HK	4	HL
2	5	GH	6	MM	7	JH	4	MS

Like Example 5, the following two-step PROC TRANSPOSE solution will produce a dataset that *looks* correct:

```
proc transpose data=have out=tall ;
  by idnum date;
  var var1-var2;
run;
```

```
proc transpose data=tall out=want (drop=_:) delimiter=_Qtr;
  by idnum;
  id _name_ date;
  var col1;
  format date qtr1.;
run;
```

Looks, however, can be deceiving. In the first step of the process, PROC TRANSPOSE was used to make the wide file long, thus putting the values for both *var1* and *var2* into a single column labeled *col1*. As such, all of the data in that column had to be of the same type and, since one of the variables was a character variable, all of the resulting transposed variables had to be converted to being character variables.

That may not pose a problem. If the transposed file is only going to be used to print a report, the PROC TRANSPOSE behavior would be quite acceptable (other than the speed, coding time, and resource utilization of course). However, the procedure's behavior would definitely pose a significant problem if the transposed file had to be used for additional analytical purposes. The following call of the %transpose macro, besides requiring less code and running faster, will not change the type of any of the variables:

```
%transpose(data=have, out=want, by=idnum, id=date, Guessingrows=1000,
           format=qtr1., delimiter=_Qtr, var=var1-var2.)
```

PROBLEM 7: Speed-Part IV

When we first started testing the idea that a datastep approach might be faster than using PROC TRANSPOSE to accomplish complex data transformations, our goal was simply to identify and document some of the processing speed benefits one could realize from the approach. We initially developed the %transpose macro simply to make it easier for us to do the various comparisons we thought might be needed to support the contention. As it turned out, the speed benefits from using a datastep approach were readily substantiated, but we realized that the code users would have to write in order to benefit from the approach might be too difficult or time consuming for many SAS users to accomplish. Then we realized that the macro, itself, could alleviate that problem.

In the various comparisons we conducted as part of the project, the time needed to sort data was never considered, as sorting was required by both approaches. However, once we realized that the macro would be the subject of the paper, we got a bit greedy and began looking for ways to optimize the discrepancy. One answer, as it turned out, was simply to force the macro to follow best practices.

In a tutorial presented at the 1996 SUGI meeting, Bruce Gilsen presented a number of things SAS programmers can do to improve the efficiency of their programs. One of those methods was to only keep the variables which will be necessary for a particular analysis. Unfortunately, the efficiency was only mentioned as the sixth out of nine methods, and was only mentioned without showing how effective the efficiency could be.

Since all of our comparisons included both a sort and a data transposition, we decided to investigate the efficiency's effect on both tasks. We were led to look at that particular efficiency because we realized that both PROC TRANSPOSE and the %transpose macro statements, options and parameters already provided all of the necessary information. For example, when you run either, you provide SAS with the knowledge of what *by*, *id*, *var* and *copy* variables are to be included in the process. No other variables are needed to produce the desired result.

Thus, if it turned out to be a significant time saver, it would be easy to incorporate a PROC SORT with a keep statement as part of the macro. In addition to improving performance, it would have the additional benefit of significantly reducing the amount of code one would have to write, thus reducing the opportunity for error.

Have you ever considered how much time you could save by excluding irrelevant data from your tasks and analyses? Even with the present paper's authors' over 100 years combined experience as SAS users, and the fact that we are all quite familiar with the technique, most of us seldomly use it. Additionally, many of us have probably wasted time running a procedure that required a sorted dataset, only to discover either that we had failed to sort the incoming data, or that the data weren't sorted because the dataset's sort flag had contained an incorrect value.

How much time is saved by dropping irrelevant variables? Presume that you had to run a PROC TRANSPOSE on a dataset like the one that would be created by the following code:

```
data have (drop=months i);
  array var(*) var1-var1000;
  do idnum=1 to 10000;
    date="01dec2010"d;
    do months=3 to 12 by 3;
      date=intnx('month',date,3);
      do i=1 to 1000;
        var(i) = ceil( 9*ranuni(123) );
      end;
      output;
    end;
  end;
run;
```

The above code creates a 40,000 record dataset that has 1,002 variables, namely *IDNUM*, *date*, and 1,000 variables labeled *var1* thru *var1000*. Given that dataset, assume that we want to transpose it to produce a file like the one shown in Example 1 on the following page; i.e., with one record for each of the 10,000 *IDNUMs*, and each record containing four variables that reflect the values of *var1* for each of the four quarters (we didn't think you wanted to see the results for all 10,000 idnums so we only showed the first two).

Example 9

idnum	var1_Qtr1	var1_Qtr2	var1_Qtr3	var1_Qtr4
1	7	8	3	7
2	6	6	5	4

Many SAS users would use code like the following in order to accomplish the task:

```
proc sort data=have out=need;
  by idnum date;
run;

proc transpose data=need out=want (drop=_:) prefix=var1_Qtr;
  by idnum;
  id date;
  var var1;
  format date Qtr1.;
run;
```

The only variables that PROC TRANSPOSE uses are those that are specified in the *by, id, var* and *copy* statements. Dropping any unnecessary variables when running PROC SORT will result in dramatically improving the performance of your run. And, while not quite as dramatic, it will also have a significant performance improvement effect on your running PROC TRANSPOSE.

The following code would have run significantly faster:

```
proc sort data=have (keep=idnum date var1) out=need noequals;
  by idnum date;
run;

proc transpose data=need out=want (drop=_:) prefix=var1_Qtr;
  by idnum;
  id date;
  var var1;
  format date Qtr1.;
run;
```

We tested the differences between the two methods, and between those methods and the macro described in the present paper, by running 50 trials of each of the two conditions, randomly selecting which method would be run on each trial. Bruce Gilsen's efficiency tip allowed PROC SORT to run 7 times faster and PROC TRANSPOSE to run 4.67 times faster. As a result, we expanded the macro so that sorting and processing could both be accomplished in one call. The following call of the %transpose macro, besides requiring less code, ran the actual transposition 7 times faster than the optimized PROC TRANSPOSE method: and 32 times faster than the non-optimized PROC TRANSPOSE method:

```
%transpose(data=have, out=want, var=var1, by=idnum, id=date,
 format=Qtr1., delimiter=_Qtr, sort=yes, guessingrows=4)
```

In short, the effects of optimizing one's code cannot be overemphasized. And, since we hadn't incorporated the keep option in the macro when we ran the previous tests, all of the comparisons documented earlier in this paper are clearly significant underestimates of the performance gains that can actually be achieved. Further, this test showed that the beneficial effects the macro can have aren't limited to just *big data*.

And, the use of a keep statement is not the only means one has available in order to optimize efficiency when using PROC SORT. In a 2001 SUGI tutorial, Philip Mason suggested that the order of records within by groups seldom has to be maintained, yet the order will be maintained unless the PROC SORT *noequals* option is specified. Philip stated that he has typically saved approximately five percent of his processing time by including the option. Since order within by groups is totally irrelevant when one is using PROC TRANSPOSE, we evaluated the option's effect on file sizes of 250,000 and 2.5 million records, when there were three variables and only one *by* variable. The savings were 12.5 percent on the smaller file and over 14 percent on the larger file.

Philip also suggested that using the *tagsort* option could save additional processing time, as could using views and compressed datasets, but that such use could also be costly dependent upon the size of one's data and the number of keys specified in a by statement. As such, we decided to only include the *noequals* option, but allow users to specify additional options in a *sort_options* parameter.

THE %TRANSPOSE MACRO

The %transpose macro was designed to accomplish complex data transposition tasks, quickly, and requiring less code and system resources than PROC TRANSPOSE. Basically, the program creates and runs the code that would be needed to accomplish the task using a datastep approach. The macro's named parameters can be included to ensure that the above mentioned efficiency methods will always be used, pre-sorting and transpositions can be accomplished in one call, and you can have more control over the inclusion and order of id value labels.

Named Parameters. Named parameters were used so that (1) default values could be assigned and (2) the various parameters would only have to be specified when values other than the default values are required. We attempted, as closely as possible, to use the same option names and statements as those used for PROC TRANSPOSE. When calling the macro, the default values will be used unless you specify the desired value. Thus, if you wanted the macro to typically get your data from a libname called mydata, you would modify the parameter by specifying it in the macro declaration. Example:

```
%macro transpose(libname_in=mydata, libname_out=, data=, out=, by=, id=,
   var=, autovars=, prefix=, var_first=, format=, delimiter=, copy=, drop=,
```

```
sort=, sort_options=, use_varname=, preloadfmt=, guessingrows=);
```

libname_in is the parameter to which you would assign the name of the SAS library that contains the dataset you want to transpose. If left null, and the *data* parameter is only assigned a one-level filename, the macro code will set this parameter to equal *WORK*.

libname_out is the to which you would assign the name of the SAS library where you want the transposed file written. If left null, and the out parameter only has a one-level filename assigned, the macro code will set this parameter to equal WORK

data is the parameter to which you would assign the name of the file that you want to transpose. Like with PROC TRANSPOSE, you can use either a one or two-level filename. If you assign a two-level file name, the first level will take precedence over the value set in the *libname_in* parameter. If you assign a one-level filename, the libname in the *libname_in* parameter will be used.

out is the parameter to which you would assign the name of the transposed file that you want the macro to create. Like with PROC TRANSPOSE, you can use either a one or two-level filename. If you use assign a two-level file name, the first level will take precedence over the value set in the *libname_out* parameter. If you use a one-level filename, the libname in the *libname_out* parameter will be used.

by is the parameter to which you would assign the name of the data's *by* variable or variables. The *by* parameter is identical to the *b*y statement used in PROC TRANSPOSE, namely the identification of the variable or variables that the macro will use to form *by groups*. *By groups* define the record level of the resulting transposed file.

prefix is the parameter to which you would assign the string that you want assigned at the beginning of each variable name that will be assigned to the transposed variables the macro will create.

var is the parameter to which you would assign the name or names of the variables that you want the macro to transpose. You can either leave it at is null default value, or you can assign any combination of variable names or variable list(s) that you could assign to PROC TRANSPOSE's var statement. However, there are some significant functional differences between the macro's *var* parameter and the PROC TRANSPOSE *var* statement:

- If the var parameter has a null value the macro will select variables based on the setting of the *autovars* parameter
- Both character and numeric variables will retain their variables types in the transformed dataset
- Given a combination of *by*, *id* and *var* statements, PROC TRANSPOSE would output one record for each variable in the var list for each of the by variable(s)' values. Each record would contain a variable called _name_ that will contain the name of the var variable(s), and another new variable called col1 that will contain the values of the var variable(s). The macro, on the other hand, will output one record for each value found in the *by* parameter, with each record containing either: columns for each combination of the values of the variable identified in the *id* parameter and those identified in the *var* parameter; or, if no *id* variable is specified, columns for each of the values of the variable(s) in the *var* parameter

autovars is the parameter to which you would assign the types of variables you want automatically assigned to the var parameter in the event that the var parameter has a null value. Possible values are: NUM, CHAR or ALL. All three choices will exclude the variables identified in the *by, id* and *copy* parameters. NUM will only include numeric variables, CHAR will only include character variables, and ALL will include both numeric and character variables. If left null, the macro code will set this parameter to equal NUM.

id is the parameter to which you would assign the variable whose values you want concatenated with the *var* variable(s) selected. It functions differently than the PROC TRANSPOSE *ID* statement in that (1) only one variable can be assigned and (2) the macro will allow duplicate values to exist within a given *by* variable value. In the case of the latter, only the last values will be assigned. Like the PROC TRANSPOSE *ID* statement, the *id* variable's format will be used unless the *format* parameter is specified.

var_first is the parameter that defines whether *var* names should precede *id* values in the concatenated variable names resulting from a transposition. Possible values are YES or NO. Concatenated variables names will be constructed as follows: prefix+var *or* id+delimiter+var *or* id. If left null, the macro code will set this parameter to equal YES. Thus, in effect, the *var_first* parameter provides the same capability you would have by assigning both the *id* and automatic system variable *_NAME_*, in the desired order, as *id* variables in PROC TRANSPOSE.

format is the parameter to which you would assign the format you want applied to the id variable in the event you don't want the variable's actual format to be applied. If left null, and the variable doesn't have a format assigned, the macro code will create a format based on the variable's type and length **.**

delimiter is the parameter to which you would assign the *string* you want assigned between the *id* and *var* variable values and names in the variable name that will be assigned to the transposed variables the macro will create.

copy is the parameter to which you would assign the name or names of any variables that you want the macro to copy rather than transpose. The macro's copy parameter is different than the PROC TRANSPOSE copy statement, as it will only copy the last value found for a given *by* variable.

drop is the parameter to which you would assign the name(s) of any variables you want dropped from the output. Since only *&by*, *&copy* and transposed variables are kept, this parameter would only be used if you want to drop one or more of the &by variables.

sort is the parameter to which you would indicate whether the input dataset must first be sorted before the data is transposed. Possible values are: YES or NO. If left null, the macro code will set this parameter to equal NO:

- YES:PROC SORT will be run, using the *noequals* option, any additional options specified in the *sort_options* parameter, and a *keep* option that will only keep the input dataset's *&by, &id, &var* and *&copy* variables. The sorted dataset will be written to a temporary file that will be deleted once the transformed file has been output
- NO: The dataset specified in the *data* parameter will be read, but only the *&by, &id, &var* and *&copy* variables will be kept for processing, and a *notsorted* option will be used for the *by* variable. No change will be made to the dataset specified in the *&data* parameter

sort_options. The noequals option will be used for all sorts, but this parameter can be used to specify any other sort options you might want to use to enhance the procedure's efficiency (e.g., presorted or tagsort)

use_varname is the parameter you would use if you don't want the var names to be included in the transposed variable names. Possible values are: YES or NO. If left null, the macro code will set this parameter to equal YES

preloadfmt. If you want to predefine all possible id variable values, and the order in which those values will be assigned to the transposed variables, you can use this parameter to assign a one or two-level filename for a file you want the macro to use. The file must contain a variable that has the same name as the data file's id variable, and a 2nd variable called 'order' that will reflect the desired order. The file must have one record for every id value the macro will find in the data, although it can also contain id values that aren't present in the data. Regarding the order variable, the value 1 must be assigned to the value you want furthest left, increasing by 1 for each remaining value, and the furthest right variable must be equal to the total number of id levels. If a two-level file name is specified, the first level will take precedence over the value set in the libname_in parameter. If a one-level filename is assigned, the libname in the libname_in parameter will be used (or 'work' if the libname_in parameter is null)

guessingrows. The *guessingrows* parameter doesn't have a PROC TRANSPOSE equivalent. It was added to the macro so that an entire dataset doesn't have to be read in order to assure the correct ordering of the resulting transposed variables. If left null, the macro code will set this parameter to equal the largest integer your system will recognize and all records will be read. However, if all of the possible values of the *id* variable are present in the input dataset's first 1,000 records, and you set guessingrows to equal 1000, the concatenated variable names will be in ascending order of the *id* variable's values. We realize that some will disagree with the term "guessingrows", but it was selected to be consistent with its use in other SAS procedures and the various flavors of Window's operating systems. There is no guesswork involved! The macro parameter simply limits the maximum number of rows that will be analyzed when determining the output ordering of the *id* variable's values.

USING THE %TRANSPOSE MACRO

Like PROC TRANSPOSE, how one uses the %transpose macro is totally up to the user. Many users have discovered uses for PROC TRANSPOSE that were probably never intended by the system's developers. We don't anticipate that happening with the current macro but, honestly, we simply haven't given the possibility any thought.

The only caveat we suggest is that you specify the NOQUOTELENMAX system option, prior to running the macro, if there is any chance that the combination of your transposed variable names will exceed 262 characters in length.

HOW THE %TRANSPOSE MACRO WORKS

The principal reason why the macro runs faster than PROC TRANSPOSE is that a datastep, using arrays, is simply more efficient than the procedure. The macro creates and runs a SAS program that contains such a datastep. Since PROC TRANSPOSE's statements and options necessarily contain a declaration of all relevant variables, it was easy to write the macro such that the resulting code could always take advantage of Gilsen's (1996) suggested KEEP statement efficiency. Additionally, since a datastep approach necessarily incorporates separate arrays for character and numeric variables, transposed numeric variables aren't converted to a character type like they are by PROC TRANSPOSE when mixed variable types are declared in a multiple step use of the procedure.

A good example would be the one described, earlier, regarding Problem 6. So that you don't have to refer back to that section, we'll simply repeat the example here. Given a dataset that looks like the one shown below as dataset *Have*, and that you need a transposed dataset that looks like the one labeled *Want*, on the following page, the task would be accomplished with the call of the %transpose macro as shown below.

Dataset Have

idnum	date	var1	var2
1	31MAR2013	1	SD
1	30JUN2013	2	EF
1	30SEP2013	3	HK
1	31DEC2013	4	HL
2	31MAR2013	5	GH
2	30JUN2013	6	MM
2	30SEP2013	7	JH
2	31DEC2013	4	MS

Dataset Want

idnum	var1_Qtr1	var2_Qtr1	var1_Qtr2	var2_Qtr2	var1_Qtr3	var2_Qtr3	var1_Qtr4	var2_Qtr4
1	1	SD	2	EF	3	HK	4	HL
2	5	GH	6	MM	7	JH	4	MS

```
%transpose(data=have, out=want, by=idnum, id=date,
           format=qtr1., delimiter=_Qtr, var=var1-var2)
```

The macro would create and run a datastep like the one shown below:

```
data work.want;
  set work.have (keep=idnum date var2 var1);
  by idnum notsorted;
  array want_char(*) $ var2_Qtr1 var2_Qtr2 var2_Qtr3 var2_Qtr4;
  array have_char(*) $ var2;
  format var1_: 1. var 2_: $2.;
  retain want_char;
```

```
  if first.idnum then call missing(of want_char(*));
  ___nchar=put(date,labelfmt.)*dim(have_char);
  do ___i=1 to dim(have_char);
    want_char(___nchar+___i)=have_char(___i);
  end;
  array want_num(*) var1_Qtr1 var1_Qtr2 var1_Qtr3 var1_Qtr4;
  array have_num(*) var1;
  retain want_num;
  if first.idnum then call missing(of want_num(*));
  ___nnum=put(date,labelfmt.)*dim(have_num);
  do ___i=1 to dim(have_num);
    want_num(___nnum+___i)=have_num(___i);
  end;
  drop date ___: var1-var2;
  if last.idnum then output;
run;
```

The datastep simply: (1) creates a file called *want*; (2) sets the dataset *have* (only keeping the four variables defined in *the by, id* and *var* parameters); (3) captures all of the character variables in an array called *have_char*; (4) captures all of the numeric variables in an array called *have_num*; (5) assigns the requested transposed character variables to an array called *want_char*; (6) assigns the requested transposed numeric variables to an array called *want_num*; and (7) only outputs one record for each level of the by variable(s), dropping any variables that are no longer needed.

The code required to accomplish that task, we think, is fairly self-explanatory, as documented in the full macro which is shown in Appendix I.

WHERE TO GET THE MACRO

We did our best to only include carefully written and tested code, but the code will likely have to be updated from time to time to correct for errors or enhancements that we or others might discover. Additionally, while a copy of the macro is included as an appendix to this paper, copying and pasting from a pdf file often introduces stylish looking quotation marks which aren't correctly recognized by SAS. As such, we created a page for the paper on sasCommunity.org. The page includes copies of the source code and updated versions of this paper. The page can be found at: http://www.sascommunity.org/wiki/A_Better_Way_to_Flip_(Transpose)_a_SAS_Dataset

CONCLUSION

The purpose of the present paper was to describe and share a SAS macro that could be used to accomplish complex data transpositions faster and easier than can be accomplished with PROC TRANSPOSE. While the paper's authors are rather biased evaluators in this regard, we believe that the project's goals were not only accomplished, but significantly exceeded our original goals.

Indeed, the macro allows users to complete most data transpositions faster, with less code, using fewer system resources, and we believe that the macro's learning curve won't be as steep as the one required to learn PROC TRANSPOSE. Additionally, the macro can easily be generalized to serve as a template for creating similar macros for a large number of other SAS procedures, specifically any which require the data to be sorted and/or any whose specifications define the only variables that need to be addressed. In fact, if you decide to write such a macro, please post it on this paper's sasCommunity.org page.

Of course, a user written SAS macro can't have all of the various error checking and failsafe mechanisms that actual SAS developers can achieve writing at the source code level. As such, the paper's authors have submitted a SAS Ballot item suggesting that all relevant procedures be enhanced to incorporate the same benefits that the current macro affords for most PROC TRANSPOSE tasks.

DISCLAIMER

The contents of this paper are the work of the authors and do not necessarily represent the opinions, practices or recommendations of the authors' organizations. The code presented in this paper is not intended to be a substitute

for PROC TRANSPOSE and is provided *as is without* warranty of any kind, either express or implied including, but not limited to, the implied warrantees of merchantability, fitness for a particular purpose, or non-infringement. The authors shall not be liable whatsoever for any damages arising out of the use of this paper or code, including any direct, indirect, or consequential damages.

ACKNOWLEDGMENTS

The authors are especially thankful for Art Carpenter's and Mary Rosenbloom's contributions in reviewing the present paper. Their suggestions were invaluable in formalizing both the macro and the present paper.

CONTACT INFORMATION

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

Arthur Tabachneck, Ph.D., President
myQNA, Inc.
Thornhill, ON Canada
E-mail: atabachneck@gmail.com

Xia Ke Shan
Chinese Financial Electrical Company
Beijing, China
E-mail: keshan.xia@gmail.com

Robert Virgile
Robert Virgile Associates, Inc.
Lexington, MA
E-mail: rvirgile@verizon.net

Joe Whitehurst
High Impact Technologies
Atlanta GA
E-mail: joewhitehurst@gmail.com

REFERENCES

Base SAS® Help and Documentation, SAS Institute Inc. 2009, Cary, NC, USA, http://support.sas.com/documentation/onlinedoc/base/

SAS Program Efficiency for Beginners, Gilsen, Bruce, SUGI 1996 Proceedings, http://www.sascommunity.org/sugi/SUGI96/Sugi-96-53%20Gilsen.pdf

SAS Tips I Learnt While at Oxford, Mason, Philip, SUGI 2001 Proceedings, http://www2.sas.com/proceedings/sugi26/p020-26.pdf

APPENDIX I

/*THE %TRANSPOSE MACRO*/

```
/** The %transpose macro
  *
  * This program performs transpositions of SAS datasets very similar to those that
  * can be achieved with PROC TRANSPOSE, but in such a manner that is easier to use
  * when performing complex transpositions and runs significantly faster
  *
  * AUTHORS: Arthur Tabachneck, Xia Ke Shan, Robert Virgile and Joe Whitehurst
  * CREATED: January 20, 2013
  * MODIFIED: May 2, 2013 (Added use_varname parameter)
  * MODIFIED: May 8, 2013 (Moved sort to before GuessingRows and added code to
                           keep formats of transposed variables)
  * MODIFIED: May 10, 2013 (Corrected errors that were discovered in code)
  * MODIFIED: May 11, 2013 (Default Guessingrows parameter set to highest integer)
  * MODIFIED: May 12, 2013 (Preloadfmt parameter added)

  Parameter Descriptions:

  *libname_in* the parameter to which you can assign the name of the SAS library that
  contains the dataset you want to transpose. If left null, and the data parameter is
  only assigned a one-level filename, the macro will set this parameter to equal WORK

  *libname_out* the parameter to which you can assign the name of the SAS library
  where you want the transposed file written. If left null, and the out parameter only
  has a one-level filename assigned, the macro will set this parameter to equal WORK

  *data* the parameter to which you would assign the name of the file that you want
  to transpose.  Like with PROC TRANSPOSE, you can use either a one or two-level
  filename.  If you assign a two-level file name, the first level will take precedence
  over the value set in the libname_in parameter.  If you assign a one-level
  filename, the libname in the libname_in parameter will be used

  *out* the parameter to which you would assign the name of the transposed file that
  you want the macro to create.  Like with PROC TRANSPOSE, you can use either a one
  or two-level filename.  If you use assign a two-level file name, the first level
  will take precedence over the value set in the libname_out parameter.  If you use a
  one-level filename, the libname in the libname_out parameter will be used

  *by* the parameter to which you would assign the name of the data's by variable
  or variables.  The by parameter is identical to the by statement used in PROC
  TRANSPOSE, namely the identification of the variable that the macro will use to
  form by groups.  By groups define the record level of the resulting transposed file

  *prefix* the parameter to which you would assign a string that you want each
  transposed variable to begin with

  *var* the parameter to which you would assign the name or names of the variables
  that you want the macro to transpose.  You can assign any combination of variable
  names or variable list(s) that you could assign to PROC TRANSPOSE's var statement.
  If left null, all variables, all numeric variables, or all character variables
  (other than by, id and copy variables) will be assigned, dependent upon the value
  assigned to the autovars parameter

  *autovars* the parameter to which you would assign the types of variables you want
  automatically assigned to the var parameter in the event that the var parameter has
  a null value.  Possible values are: NUM, CHAR or ALL.  If left null, the macro code
  will set this parameter to equal NUM

  *id* the parameter to which you would assign the variable whose values you want
  concatenated with the var variable(s) selected.  Only one variable can be assigned

  *var_first* the parameter that defines whether var names should precede id values in
  the concatenated variable names resulting from a transposition. Possible values are
```

```
  YES or NO.  Concatenated variables names will be constructed as follows:
  prefix+var or id+delimiter+var or id.  If left null, the macro code will set this
  parameter to equal YES

  *format* the parameter to which you would assign the format you want applied
  to the id variable in the event you don't want the variable's actual format
  to be applied. If left null, and the variable doesn't have a format assigned,
  the macro code will create a format based on the variable's type and length

  *delimiter* the parameter to which you would assign the string you want
  assigned between the id values and var variable names in the variable name
  that will be assigned to the transposed variables the macro will create

  *copy* the parameter to which you would assign the name or names of any
  variables that you want the macro to copy rather than transpose

  *drop* the parameter to which you would assign the name(s)  of any variables
  you want dropped from the output.  Since only &by, &copy and transposed variables
  are kept, this parameter would only be used if you want to drop one or more of the
  &by variables

  *sort* the parameter to which you would indicate whether the input dataset
  must first be sorted before the data is transposed.  Possible values are:
  YES or NO. If left null, the macro code will set this parameter to equal NO

  *sort_options* while the noequals option will be used for all sorts, you would use
  this parameter to specify any additional options you want used (e.g., presorted,
  force and/or tagsort

  *use_varname* the parameter you would use if you don't want the var names
  to be included in the transposed variable names.  Possible values are: YES or NO.
  If left null, the macro code will set this parameter to equal YES

  *preloadfmt* If you want to predefine all possible id variable values, and the
  order in which those values will be assigned to the transposed variables, you can
  use this parameter to assign a one or two-level filename for a file you want the
  macro to use.  The file must contain a variable that has the same name as the data
  file's id variable, and a 2nd variable called 'order' that will reflect the desired
  order. The file must have one record for every id value the macro will find in the
  data, although it can also contain id values that aren't present in the data.
  Regarding the order variable, the value 1 must be assigned to the value you want
  furthest left, increasing by 1 for each remaining value, and the furthest right
  variable must be equal to the total number of id levels. If a two-level file name
  is specified, the first level will take precedence over the value set in the
  libname_in parameter. If a one-level filename is assigned, the libname in the
  libname_in parameter will be used (or 'work' if the libname_in parameter is null)

  Example:
  data order;
    input date date9. order;
    cards;
  31mar2013 1
  30jun2013 2
  30sep2013 3
  31dec2013 4
  ;

  *guessingrows* the parameter you would use to specify the maximum number of rows
  that will be read to determine the output ordering of the id variable's values.
  If left null, the macro will set this parameter high enough to read all records

*/

%macro transpose(libname_in=,
                 libname_out=,
                 data=,
                 out=,
```

```
                  by=,
                  prefix=,
                  var=,
                  autovars=,
                  id=,
                  var_first=,
                  format=,
                  delimiter=,
                  copy=,
                  drop=,
                  sort=,
                  sort_options=,
                  use_varname=,
                  preloadfmt=,
                  guessingrows=);

/*Check whether the data and out parameters contain one or two-level filenames*/
  %if %sysfunc(countw(&data.)) eq 2 %then %do;
    %let libname_in=%scan(&data.,1);
    %let data=%scan(&data.,2);
  %end;
  %else %if %length(&libname_in.) eq 0 %then %do;
    %let libname_in=work;
  %end;

  %if %sysfunc(countw(&out.)) eq 2 %then %do;
    %let libname_out=%scan(&out.,1);
    %let out=%scan(&out.,2);
  %end;
  %else %if %length(&libname_out.) eq 0 %then %do;
    %let libname_out=work;
  %end;

  /*obtain last by variable*/
  %if %length(&by.) gt 0 %then %do;
    %let lastby=%scan(&by.,-1);
  %end;
  %else %do;
    %let lastby=;
  %end;

/*Ensure a format is assigned to an id variable*/
  %if %length(&id.) gt 0 %then %do;
    proc sql noprint;
      select type,length,%sysfunc(strip(format))
        into :tr_macro_type, :tr_macro_len, :tr_macro_format
          from dictionary.columns
            where libname="%sysfunc(upcase(&libname_in.))" and
                  memname="%sysfunc(upcase(&data.))" and
                  upcase(name)="%sysfunc(upcase(&id.))"
        ;
    quit;

    %if %length(&format.) eq 0 %then %do;
      %if &tr_macro_format. ne %then %do;
        %let format=&tr_macro_format.;
      %end;
      %else %if "&tr_macro_type." eq "num " %then %do;
        %let format=%sysfunc(catt(best,&tr_macro_len.,.));
      %end;
      %else %do;
        %let format=%sysfunc(catt($,&tr_macro_len.,.));
      %end;
    %end;
  %end;

/*Create macro variable to contain a list of variables to be copied*/
```

```
 %let to_copy=;
  %if %length(&copy.) gt 0 %then %do;
    data _temp;
      set &libname_in..&data. (obs=1 keep=&copy.);
    run;

    proc sql noprint;
      select name
        into :to_copy separated by " "
          from dictionary.columns
            where libname="WORK" and
                  memname="_TEMP"
        ;
      quit;
  %end;

/*Populate var parameter in the event it has a null value*/
  %if %length(&var.) eq 0 %then %do;
    data _temp;
      set &libname_in..&data. (obs=1 drop=&by. &id. &copy.);
    run;

    proc sql noprint;
      select name
        into :var separated by " "
          from dictionary.columns
            where libname="WORK" and
                  memname="_TEMP"
        %if %sysfunc(upcase("&autovars.")) eq "CHAR" %then %do;
                  and type="char"
        %end;
        %else %if %sysfunc(upcase("&autovars.")) ne "ALL" %then %do;
                  and type="num"
        %end;
        ;
      quit;
  %end;

/*Ensure guessingrows parameter contains a value*/
  %if %length(&guessingrows.) eq 0 %then %do;
    %let guessingrows=%sysfunc(constant(EXACTINT));
  %end;

/*Initialize macro variables*/
  %let vars_char=;
  %let varlist_char=;
  %let vars_num=;
  %let varlist_num=;
  %let formats_char=;
  %let format_char=;
  %let formats_num=;
  %let format_num=;

/*Create file _temp to contain one record with all var variables*/
  data _temp;
    set &libname_in..&data. (obs=1 keep=&var.);
  run;

/*Create macro variables containing untransposed var names and formats*/
  proc sql noprint;
    select name, case
                   when missing(format) then " $"||strip(put(length,5.))||'.'
                   else strip(format)
                 end
      into :vars_char separated by " ",
           :formats_char separated by "~"
        from dictionary.columns
```

```
          where libname="WORK" and
                memname="_TEMP" and
                type="char"
    ;
    select name, case
                   when missing(format) then "best12."
                   else strip(format)
                 end
      into :vars_num separated by " ",
           :formats_num separated by "~"
        from dictionary.columns
          where libname="WORK" and
                memname="_TEMP" and
                type="num"
    ;
  quit;

/*If sort parameter has a value of YES, create a sorted temporary data file*/
  %if %sysfunc(upcase("&sort.")) eq "YES" %then %do;
    %let notsorted=;
    proc sort data=&libname_in..&data.
                (keep=&by. &id. &vars_char. &vars_num. &to_copy.)
                 out=_temp &sort_options. noequals;
      by &by.;
    run;
    %let data=_temp;
    %let libname_in=work;
  %end;
  %else %do;
    %let notsorted=notsorted;
  %end;

/*Create macro variables containing ordered lists of the requested transposed variable
  names for character (varlist_char) and numeric (varlist_num) var variables */
  %if %length(&preloadfmt.) gt 0 %then %do;
    %if %sysfunc(countw(&preloadfmt.)) eq 1 %then %do;
      %let preloadfmt=&libname_in..&preloadfmt.;
    %end;
  %end;

  proc sql noprint;
  %do i=1 %to 2;
    %if &i. eq 1 %then %let i_type=char;
    %else %let i_type=num;
    %if %length(&&vars_&i_type.) gt 0 %then %do;
    select distinct
      %do j=1 %to 2;
        %if &j. eq 1 %then %let j_type=;
        %else %let j_type=format;
        %do k=1 %to %sysfunc(countw(&&vars_&i_type.));
      "&j_type. "||"&prefix."||
          %if %sysfunc(upcase("&var_first.")) eq "NO" %then %do;
        strip(put(&id.,&format))||"&delimiter."
            %if &k. lt %sysfunc(countw(&&vars_&i_type.)) %then ||;
            %if %sysfunc(upcase("&use_varname.")) ne "NO" %then %do;
              %if &k. ge %sysfunc(countw(&&vars_&i_type.)) %then ||;
        strip(scan("&&vars_&i_type.",&k.))
            %if &k. lt %sysfunc(countw(&&vars_&i_type.)) %then ||;
          %end;
        %end;
        %else %do;
          %if %sysfunc(upcase("&use_varname.")) ne "NO" %then
        strip(scan("&&vars_&i_type.",&k.))||;
        "&delimiter."||strip(put(&id.,&format))
        %end;
        %if &j. eq 2 %then
        ||" "||strip(scan("&&formats_&i_type.",&k.,"~"))||";";
```

```
          %if &k. lt %sysfunc(countw(&&vars_&i_type.)) %then ||;
          %else ,;
        %end;
      %end;
      %if "&tr_macro_type." eq "num " %then &id. format=best12.;
        %else &id.;
        %if %length(&preloadfmt.) gt 0 %then ,order;
          into :varlist_&i_type. separated by " ",
               :format_&i_type. separated by " ",
               :idlist separated by " "
           %if %length(&preloadfmt.) gt 0 %then %do;
               ,:idorder separated by " "
            from &preloadfmt.
            order by order
           %end;
           %else %do;
            from &libname_in..&data. (obs=&guessingrows. keep=&id.)
            order by &id.
           %end;
    ;
    %let num_numlabels=&sqlobs.;
    %end;
  %end;
  quit;

/*Create a format that will be used to assign values to the transposed variables*/
  %if %length(&preloadfmt.) eq 0 %then %do;
   data _for_format;
   %do i=1 %to &num_numlabels.;
    start=%sysfunc(quote(%scan(&idlist,&i)));
    output;
   %end;
   run;
  %end;

  data _for_format;
    %if %length(&preloadfmt.) gt 0 %then set &preloadfmt. (rename=(&id.=start));
    %else set _for_format;
    ;
    %if "&tr_macro_type." eq "num " %then retain fmtname "labelfmt" type "N";
    %else retain fmtname "$labelfmt" type "C";
    ;
    label=
     %if %length(&preloadfmt.) eq 0 %then _n_-1;
     %else order-1;
     ;
  run;

  proc format cntlin = _for_format;
  run ;

/*Create and run the datastep that does the transposition*/
  data &libname_out..&out.;
    set &libname_in..&data. (keep=&by. &id.
      %do i=1 %to %sysfunc(countw("&vars_char."));
        %scan(&vars_char.,&i.)
      %end;
      %do i=1 %to %sysfunc(countw("&vars_num."));
        %scan(&vars_num.,&i.)
      %end;
      %do i=1 %to %sysfunc(countw("&to_copy."));
        %scan(&to_copy.,&i.)
      %end;
      );
    by &by. &notsorted.;
    &format_char. &format_num.
  %if %length(&vars_char.) gt 0 %then %do;
```

```
    array want_char(*) $
    %do i=1 %to %eval(&num_numlabels.*%sysfunc(countw("&vars_char.")));
      %scan(&varlist_char.,&i.)
    %end;
    ;
    array have_char(*) $ &vars_char.;
    retain want_char;;
    if first.&lastby. then call missing(of want_char(*));
    ___nchar=put(&id.,labelfmt.)*dim(have_char);
    do ___i=1 to dim(have_char);
      want_char(___nchar+___i)=have_char(___i);
    end;
  %end;
  %if %length(&vars_num.) gt 0 %then %do;
    array want_num(*)
    %do i=1 %to %eval(&num_numlabels.*%sysfunc(countw("&vars_num.")));
      %scan(&varlist_num.,&i.)
    %end;
    ;
    array have_num(*) &vars_num.;
    retain want_num;
    if first.&lastby. then call missing(of want_num(*));
    ___nnum=put(&id.,labelfmt.)*dim(have_num);
    do ___i=1 to dim(have_num);
      want_num(___nnum+___i)=have_num(___i);
    end;
  %end;
    drop &id. ___: &var. &drop.;
    if last.&lastby. then output;
  run;

/*Delete all temporary files*/
  proc delete data=work._temp work._for_format;
  run;

%mend transpose;
options NOQUOTELENMAX;
```