Paper 2419-2018

An Easier and Faster Way to Untranspose a Wide File

Arthur S. Tabachneck, Ph.D., AnalystFinder, Inc.; Matthew Kastin, NORC at the University of Chicago; Joe Matise, NORC at the University of Chicago; Gerhard Svolba, Ph.D., SAS Institute, Inc.

ABSTRACT

PROC TRANSPOSE is an extremely powerful tool for making long files wide, and wide files less wide or long, but getting it to do what you need often involves a lot of time, effort, and a substantial knowledge of SAS® functions and data step processing. This is especially true when you have to untranspose a wide file that contains both character and numeric variables. And, while the procedure usually seamlessly handles variable types, lengths and formats, it doesn't always do that and only creates a system variable (i.e., _label_) to capture variable labels. The present paper introduces a macro that simplifies the process, significantly reduces the amount of coding and programming skills needed (thus reducing the likelihood of producing the wrong result), runs up to 50 or more times faster than the multiple PROC TRANSPOSE and data steps that would otherwise be needed, and either creates untransposed variables that inherit all of the original variables' characteristics or creates a file that contains all of the relevant metadata.

INTRODUCTION

We wrote the %UNTRANSPOSE macro after seeing a post on the SAS Community Forum where the poster needed to make an extremely wide file less wide, but keep all of the variable formats, lengths and labels that were present in the wide file. The file had a combination of numeric and character variables. Throughout this paper we use the term *untranspose* to mean changing a file from being wide to being either long or less wide.

THE PROBLEM

If you've ever had to rearrange a transposed SAS dataset, converting it from being a wide dataset back to the long or less wide dataset that was used to create it, you're probably already familiar with PROC TRANSPOSE. If you have never been 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 on the following page, and you wanted to create a file like the one shown in Example 2 (i.e., untranspose the dataset so that it has separate records for each ID and year combination):

```
data have;
  input id income2015-income2017
           expenses2015-expenses2017
         (debt2015-debt2017) ($);
  label income2015='Household Income'
        income2016='Household Income'
        income2017='Household Income'
        expenses2015='Household Expenses'
        expenses2016='Household Expenses'
        expenses2017='Household Expenses'
        debt2015='Household Debt'
        debt2016='Household Debt'
        debt2017='Household Debt';
  cards;
1 70000 75500 80000 60000 70000 81000 no no yes
2 50000 52000 55000 42000 53000 60000 no yes yes
3 80000 90000 99000 70000 75000 85000 no  no  no
;
```

Example 1

Obs	id	income2015	income2016	income2017	expenses2015	expenses2016	expenses2017	debt2015	debt2016	debt2017
1	1	70000	75500	80000	60000	70000	81000	no	no	yes
2	2	50000	52000	55000	42000	53000	60000	no	yes	yes
3	3	80000	90000	99000	70000	75000	85000	no	no	no

Example 2

Obs	id	income	expenses	debt	year
1	1	70000	60000	no	2015
2	1	75500	70000	no	2016
3	1	80000	81000	yes	2017
4	2	50000	42000	no	2015
5	2	52000	53000	yes	2016
6	2	55000	60000	yes	2017
7	3	80000	70000	no	2015
8	3	90000	75000	no	2016
9	3	99000	85000	no	2017

With code like the following you could use PROC TRANSPOSE to expand the file so that it had one record for each id/year combination, and maintain each variable's length, format, and labels:

```
proc transpose data=have out=longi prefix=income;
  by id;  var income2015-income2017; run;

data _null_;
  set longi (obs=1);
  call execute('proc datasets library=work nolist;modify longi;');
  forexec=catt("label income1='",_label_,"';quit;");  call execute(forexec);
run;

proc transpose data=have out=longe prefix=expenses;
  by id;  var expenses2015-expenses2017; run;

data _null_;
  set longe (obs=1);
  call execute('proc datasets library=work nolist;modify longe;');
  forexec=catt("label expenses1='",_label_,"';quit;"); call execute(forexec);
run;

proc transpose data=have out=longd prefix=debt;
  by id;  var debt2015-debt2017; run;

data _null_;
  set longd (obs=1);
  call execute('proc datasets library=work nolist;modify longd;');
  forexec=catt("label debt1='",_label_,"';quit;"); call execute(forexec);
run;

data want (drop=_:);
  set longi (rename=(income1=income) drop=_:);
  set longe (rename=(expenses1=expenses) drop=_:);
  set longd (rename=(debt1=debt));
  year=input(substr(_name_, 5), 4.);
run;
```

PROBLEM 1: COMPLEXITY

While that appears simple enough, and produces a usable result, the task can quickly become far more complex than initially meets the eye. If the wide table had captured 100 variables over different points in time, then the task would require 100 separate PROC TRANSPOSE calls, each creating uniquely named output files and untransposing one specific variable. Then, like the code in the above example, a data step would be needed for one of the files in order to extract the ID values, as well as one to merge the 100 outputs and rename the untransposed variables.

Since transposed variable names can include a prefix, a variable name, a delimiter, either fixed or variable length character or numeric ID values, and a suffix, the task of extracting the ID values can quickly become quite complex..

Some years ago one of the current paper's authors realized this and offered the MAKELONG macro to the SAS community (Svolba, G. 2008, 2014). Dr. Svolba's macro was definitely a step in the right direction, as it ran the necessary PROC TRANSPOSE and merge steps without requiring the user to do such things as redundantly type variable names, assign unique names for all of the temporary files that might have to be created, write code to extract ID values from the transposed variable names, or create a data step to merge the resulting files. Dr. Svolba's macro did everything that was needed as long as the input and ID variables were all numeric, and the wide file's transposed variable names didn't include such strings as prefixes, delimiters, and suffixes.

PROBLEM 2: OPTIMAL DESIGN

The simplest solution, of course, would be to follow Dr. Svolba's lead and refine his MAKELONG macro to account for both character and numeric variables, as well as include code to parse ID values from variable names that might include any combination of prefixes, variable names, delimiters, ID values, and suffixes. However, based on a paper co-written by another of the current paper's authors (Tabachneck, Ke Shan, Virgile, and Whitehurst, 2013), we knew that it would be more efficient to complete the entire process using a single data step instead of just simplifying the code needed to run PROC TRANSPOSE. The %transpose macro offered in the Tabachneck *et.al.* paper makes wide files wider, and completes such transpositions up to 50 or more times faster than it would take using PROC TRANSPOSE.

Another design aspect was to use named parameters that were identical to PROC TRANSPOSE's statements and options. Our goal, in that regard, was to reduce if not eliminate the learning curve needed to run the macro by any user who was already familiar with PROC TRANSPOSE.

As an example, the code needed to untranspose the dataset shown in Example 1 would be:

```
%untranspose(data=have,out=want,by=id,id=year,var=income expenses debt)
```

In building our test models, we discovered a minor inefficiency in using the PROC TRANSPOSE method, namely that the procedure will output a record even if the record only has non-missing values for the by and ID variables. As such, we included a parameter called *missing* so that the user can control whether such records will be output.

Another inefficiency of PROC TRANSPOSE is that when it's used to untranspose a dataset from being wide to being long, it creates a field called _label_ to capture the variable labels. However, it adds that variable to every record that is output. Additionally, during that process, other metadata (like length, format, type and informat) are lost if one is untransposing a combination of character and numeric variables. As such, the %untranspose macro lets the user specify a file to which they'd like such metadata written.

Finally, when PROC TRANSPOSE is used to untranspose a dataset from being wide to being long, and the dataset includes a combination of numeric and character variables, the numeric variables are output as right justified character variables, the length of the new variable is set as the widest variable included in the process regardless of whether any of the fields require that many characters, and formatted variables are output in their formatted form without giving the user any way to identify the original values. Our solution was to left justify such variables, give users the ability to set the maximum width of the resulting variables, output the actual values, and provide the user with a parameter to indicate that they want the macro to create a file containing all of the relevant metadata.

PROBLEM 3: SPEED

The %UNTRANSPOSE macro was designed to reduce the amount of time users would have to spend writing the code needed to untranspose wide datasets. However, equally if not more important, the macro will complete data untranspositions between 2 to 100 or more times faster than it would take using PROC TRANSPOSE.

THE %UNTRANSPOSE MACRO

The %untranspose macro was designed to complete complex data untransposition tasks quicker than accomplishing the same task using PROC TRANSPOSE and, at the same time, require less code and system resources than PROC TRANSPOSE. Basically, the program creates and runs the code that is needed to accomplish the task using a datastep. The macro's various named parameters can be included to match any configuration of variable naming conventions that PROC TRANSPOSE allows, as well as some that haven't yet been addressed by the procedure.

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 untranspose(libname_in=mydata,
                   libname_out=,
                   data=,
                   out=,
                   by=,
                   prefix=,
                   var=,
                   id=,
                   id_informat=8.,
                   id_format=8.,
                   var_first=yes,
                   delimiter=,
                   suffix=,
                   copy=,
                   missing=NO,
                   metadata=,
                   makelong=,
                   max_length=);
```

libname_in (NOT REQUIRED) is the parameter to which you can assign the name of the SAS library that contains the dataset you want to untranspose. 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 (NOT REQUIRED) is the parameter to which you can assign the name of the SAS library where you want the untransposed 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 (REQUIRED) is the parameter to which you would assign the name of the file that you want to untranspose. 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. Additionally, as with PROC TRANSPOSE, the data parameter will also accept data step options. Thus, for example, if you had a dataset called 'have' and want to limit the untransposition to just the first 10 records, you could specify it as: data=have (obs=10). Any data step options accepted by a SAS data step can be included

out (REQUIRED) is the parameter to which you would assign the name of the untransposed 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 (ONLY NECESSARY IF YOU HAVE A BY VARIABLE) is the parameter to which you would assign the name of the dataset's by variable or variables. The parameter is identical to the by statement used in PROC TRANSPOSE, namely the identification of the variable (if any) that had been used to form by groups. By groups define the record level of the wide file you want to create

prefix (ONLY NECESSARY IF YOUR TRANSPOSED VARIABLE NAME(S) BEGIN WITH A PREFIX) is the parameter to which you would assign the string (if any) that each transposed variable begins with

var (REQUIRED) is the parameter to which you would assign the name or names of the original untransposed variables

id (ONLY NECESSARY IF YOUR TRANSPOSED VARIABLE NAME(S) CONTAIN ID VALUES) is the parameter to which you specify the variable name that was used as the ID variable when the transposed file was created. Only one variable can be assigned

id_informat (ONLY NECESSARY IF 8. SHOULD NOT BE USED AS THE INFORMAT FOR EXTRACTING ID VALUES) is the parameter to which you would assign the informat used to extract the id variable's values

id_format (ONLY NECESSARY IF 8. SHOULD NOT BE ASSIGNED AS THE FORMAT FOR EXTRACTED ID VALUES) is the parameter to which you would indicate the format you want assigned to the id variable

var_first (ONLY NECESSARY IF ID VALUES PRECEDE VARIABLE NAMES IN THE TRANSPOSED

VARIABLE NAMES or IF THE TRANSPOSED VARIABLE NAME(S) DON'T INCLUDE THE VARIABLE NAME) is the parameter that defines whether var names precede id values in the transposed variable names. Possible values are YES, NO or N/A and must be correctly assigned to reflect the way the transposed variables were formed:

YES=prefix (if used) + var + delimiter (if used) + id + suffix (if used)

NO=prefix (if used) + id + delimiter (if used) + var + suffix (if used)

N/A=prefix (if used) + id + suffix (if used)

delimiter (ONLY NECESSARY IF YOUR TRANSPOSED VARIABLE NAME(S) CONTAIN A DELIMITER) is the parameter to which you assign the string (if any) that was used to separate var and ID values

suffix (ONLY NECESSARY IF YOUR TRANSPOSED VARIABLE NAME(S) END WITH A SUFFIX) is the parameter to which you assign the string (if any) that each transposed variable ends with

copy (ONLY NECESSARY IF YOUR WIDE FILE CONTAINS ONE OR MORE VARIABLES THAT SHOULD BE COPIED RATHER THAN UNTRANSPOSED) is the parameter to which you assign the name or names of any variables that had been copied rather than transposed

missing (ONLY NECESSARY IF YOU WANT TO OUTPUT A RECORD EVEN IF THE ONLY NON-MISSING VARIABLES ARE THE BY, ID OR COPY VARIABLES) PROC TRANSPOSE will output untransposed records even if the only non-missing variables are the BY, ID and COPY variables. If you want the macro to behave similarly set this parameter to equal YES

metadata (ONLY NECESSARY IF YOU WANT TO OUTPUT A FILE CONTAINING THE UNTRANSPOSED VARIABLES' METDATA) To output a file containing the untransposed variables' metadata, use this parameter with a value equal to the one or two-level dataset name that you want the macro to create

makelong (ONLY NECESSARY IF YOU WANT TO OUTPUT A SEPARATE RECORD FOR EACH BY VARIABLE, ID VALUE AND VARIABLE COMBINATION) This parameter will automatically be set to yes if no ID variable is declared with the ID parameter. If you do declare an ID variable, set this parameter to YES if you want the macro to output a long file

max_length (ONLY NECESSARY IF YOU WANT TO CONTROL THE LENGTH OF ALL UNTRANSPOSED VARIABLES) This parameter is only applicable to those cases where you are untransposing a file from being wide to being long. If used it should be used cautiously as it could result in losing data

HOW THE %UNTRANSPOSE MACRO WORKS

The principal reasons why the macro runs faster than PROC TRANSPOSE are that (1) a data step is simply more efficient than the procedure and (2) only one data step is needed to accomplish the task regardless of the number of variables involved. The macro creates and runs SAS code that contains such a data step. Additionally, where PROC TRANSPOSE only populates a system variable called _LABEL_ in the event that a parent variable has a label assigned, the macro was written to so that wherever possible each variable automatically inherits its parent's label. And, if desired, the user can output all relevant metadata to a SAS dataset.

A good example to show how the macro works would be to show how it creates Example 2 from Example 1. Given that Example 1 was a dataset called *have*, and that you wanted the desired output file (Example 2) to be called *Want*, the task would be accomplished with the call of the %transpose macro as shown below.

```
%untranspose(data=have,out=want,by=id,id=year,var=income expenses debt)
```

What the macro actually does. The macro would create and run a data step similar to the one shown below:

```
data work.want (keep=id year income expenses debt );
  set work.have;
  informat year 8. ;
  format year 8. ;
  length expenses 8 income 8 debt $ 8 ;
  expenses=expenses2015; income=income2015; debt=debt2015;
  if missing(expenses)+missing(income)+missing(debt) lt 3 then do;
    year=2015; output;
  end;
  expenses=expenses2016; income=income2016; debt=debt2016;
  if missing(expenses)+missing(income)+missing(debt) lt 3 then do;
    year=2016; output;
  end;
  expenses=expenses2017; income=income2017; debt=debt2017;
  if missing(expenses)+missing(income)+missing(debt) lt 3 then do;
    year=2017; output;
  end;
run;
```

The datastep: (1) creates a file called *want*; (2) sets the dataset *have; (3) assigns any existing formats, informats, lengths and labels;* (4) re-creates and populates the original variables that were declared in the var parameter; (5) populates the ID variable's value; and (6) outputs the records if any of the original variables weren't missing.

How the Macro Works – Specifying Input and Output Libnames and Filenames. The first section of the macro's code was designed to accomplish two distinct tasks, namely: (1) to populate the

libname_in and libname_out parameters in the event that either the data or out parameters contain two-level file names; and (2) to identify if any data set options are included in the data and out parameters and, if so, remove them from the data parameter and store them in macro variables called *dsoptions* and *odsoptions*:

```
%let lp=%sysfunc(findc(%superq(data),%str(%())); 
%if &lp. %then %do;
  %let rp=%sysfunc(findc(%superq(data),%str(%)),b));
  %let dsoptions=%qsysfunc(substrn(%nrstr(%superq(data)),&lp+1,&rp-&lp-1));
  %let data=%sysfunc(substrn(%nrstr(%superq(data)),1,%eval(&lp-1)));
%end;
%else %let dsoptions=;
%let lp=%sysfunc(findc(%superq(out),%str(%())); 
%if &lp. %then %do;
  %let rp=%sysfunc(findc(%superq(out),%str(%)),b));
  %let odsoptions=%qsysfunc(substrn(%nrstr(%superq(out)),&lp+1,&rp-&lp-1));
  %let out=%sysfunc(substrn(%nrstr(%superq(out)),1,%eval(&lp-1)));
%end;
 %else %let odsoptions=;
 %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;
```

How the Macro Works – Identifying Copy Variables. The next section of code was designed to identify if there are any variables to copy rather than transpose. The macro can accept any kind of variable list and does that by using a data step to create a one record file that only contains the variables to be copied. The temporary file, called T_E_M_P, is reused and repopulated throughout the macro. Since the variables are identified using a data step keep option, the macro will accept any SAS variable list. Once the file is created, PROC SQL is then used to access dictionary.columns to populate a macro variable, called *to_copy*, that will contain a space separated list of all of the variables that are to be copied:

```
%let to_copy=;
%if %length(&copy.) gt 0 %then %do;
  data t_e_m_p;
    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="T_E_M_P"
      ;
    quit;
%end;
```

How the Macro Works – Identifying Variables to Untranspose – Part I. The next section of code first uses a data step array to populate the temporary file with the names of the variables identified in the var parameter (i.e., the variable names that are imbedded in the transposed variable names). Since the purpose of this part of the code is to use the names to create two macro variables that will be used later in the code, and variable type (i.e., character or numeric) was irrelevant, only one array was necessary.

PROC SQL is then used to access dictionary.columns to populate the two macro variables, one containing a comma delimited list of the quoted variable names, and the other containing a statement used later in the macro to ensure that only records with some data will be output.

The macro variables are populated in descending order of the variable name lengths. That will be critical, later, to correctly match transposed and untransposed variable names. A second PROC SQL call is also made to create a macro variable that identifies the requested order of the var variables:

```
data t_e_m_p;
  array vars(*) &var.;
  output;
run;

proc sql noprint;
  select catt("'",name,"'"),
         catt('(not missing(',name,'))')
    into :vars separated by ",",
         :check separated by " or "
      from dictionary.columns
        where libname="WORK" and memname="T_E_M_P"
          order by length(name) descending
  ;

  select catt("'",name,"'")
    into :ordered_vars separated by ","
      from dictionary.columns
        where libname="WORK" and memname="T_E_M_P"
          order by varnum
  ;
quit;
```

How the Macro Works – Identifying Variables to Untranspose and their Metadata - Part II. The temporary dataset is then overwritten to only contain the transposed variable names and their relevant metadata. The same method used earlier in the code is used here again except, rather than creating macro variables, PROC SQL is used to replace the temporary dataset with one containing the names, formats, informats, labels, lengths and types of each of the transposed variables. Two macro variables are also created in this step, one to reflect whether any of the output variables are of type 'char', and another to reflect the maximum length of the output variables:

```
data t_e_m_p;
  set &libname_in..&data. (obs=1 &dsoptions.
  %if %length(&by.) gt 0 or %length(&copy) gt 0 %then drop=&by. &copy.;);
run;

proc sql noprint;
  create table t_e_m_p as
    select name,format,informat,label,length,type
       from dictionary.columns
         where libname="WORK" and
               memname="T_E_M_P"
  ;
  select min(type), max(length)
```

```
    into :mintype,:maxlength
      from dictionary.columns
        where libname="WORK" and
              memname="T_E_M_P"
  ;
quit;
```

How the Macro Works – Identifying ID Variable Values. A datastep, shown below, is then used to identify the ID variable values that are contained in the transposed variable names. To understand this section of the code you have to know how transposed variables might be formed by either PROC TRANSPOSE or the %transpose macro (Tabachneck, Ke Shan, Virgile and Whitehurst, 2013).

The macro contains a parameter for each possible component of the transposed variable names. The transposed variable names might start with a prefix, then might contain the variable name or ID value, then possibly a delimiter, then possibly followed by either the variable name or the ID value and, finally, might end with a suffix. As such, the macro has a parameter, called *var_first*, to indicate whether variable names precede or follow the ID values, or that no ID values exist in the transposed variable names. The macro uses a do loop to go through the variable names (which, as you might recall, are at this point in descending order of the lengths of the variable names), and uses one of three algorithms dependent upon the value of the *var_first* parameter.

If *var_first* equals 'yes' the loop continues through the list of variables until the variable (preceded by a prefix if one is used) matches the start of a transposed variable name. When it finds a match, it identifies the ID value (storing it in a variable called *id_value*) by using the substr function together with all of the relevant parameters to identify the starting position and number of characters of the ID value. Once the ID value has been found, the macro leaves the do loop so that no other transposed variables will be evaluated for that particular variable name. If *var_first* equal 'no' then a similar algorithm is applied, but using the reversed variable name and reversed transposed variable names, and finding the matching variables by comparing the reversed names once the suffix (if one was used) is applied to the end of the variable name.

If var_first has any value other than *yes* or *no*, the macro will assume that the transposed variable names only contain the variable names, possibly proceeded by a prefix, and might contain a suffix. In such a case, the macro will assign id values of '1' to each variable. The ID values won't be output and are only used as a dummy variable for sorting the T_E_M_P dataset.

By the time the last record in file T_E_M_P has been analyzed the file will contain a record for every transposed variable name, with each record containing the transposed variable name, the corresponding untransposed variable name, the untransposed variable's position as specified in the *var* parameter, the ID value represented by the transposed variable name, and the variable's assigned length, format, informat, label and variable type (i.e., character or numeric).

```
data t_e_m_p (drop=temp);
  set t_e_m_p;
  do var=&vars.;
    if %length(&id) gt 0 then do;
      if upcase("&var_first.") eq 'YES' then do;
        if catt(upcase("&prefix."),upcase(var))=:strip(upcase(name)) then
        do;
          id_value=substr(strip(name),%length(&prefix.)+length(strip(var))+
              %length(&delimiter)+1,
              length(strip(name))-%length(&prefix.)-length(strip(var))-
              %length(&delimiter)-%length(&suffix.));
          leave;
        end;
      end;
      else if upcase("&var_first.") eq 'N/A' then do;
```

```
        id_value=substr(strip(name),%length(&prefix.)+1,
             length(strip(name))-%length(&prefix.)-%length(&suffix.));
      end;
      else do;
        if strip(reverse(catt(upcase(var),upcase("&suffix.")))) =:
           strip(reverse(upcase(name))) then do;
          temp=reverse(substr(reverse(strip(name)),
              %length(&suffix)+length(strip(var))+%length(&delimiter)+1));
          id_value=substr(strip(temp),%length(&prefix.)+1);
          leave;
        end;
      end;
    end;
    else do;
      if catt(upcase("&prefix."),upcase(var),upcase("&suffix."))=:
          strip(upcase(name)) then do;
        id_value='1';
        leave;
      end;
    end;
  end;
  order=0;
  do temp=&ordered_vars.;
    order+1;
    if strip(upcase(var)) eq strip(upcase(temp)) then leave;
  end;
run;
```

The macro then uses PROC SORT to reorder file T_E_M_P in order of the ID values and, within each value, the order of the variables as specified in the *var* parameter.

```
proc sort data=t_e_m_p;
  by id_value order;
run;
```

All of the code mentioned thus far runs in the range of only a few milliseconds regardless of the number of variables being untransposed. At this point in the processing the T_E_M_P dataset contains all of the information necessary to create a data step to accomplish the main processing that will be needed. We used a datastep to invoke CALL EXECUTE to create such a data step. While it will require a couple of pages to describe how the datastep works it, too, will only require a couple of milliseconds to run.

Throughout this section we will refer to Example 1, the wide file that we want to make less wide, and refer to it as dataset *have*. We'll refer to the desired less wide file we want to create (Example 2) as dataset *want*. We will also assume that we called the macro with the following statement:

```
%untranspose(data=have,out=want,by=id,id=year,var=income expenses debt)
```

The temporary file T_E_M_P, at this point, will appear as shown in Example 3, below:

Example 3

name	format	informat	label	length	type	var	id_value	order
income2015			Yearly Income	8	num	income	2015	1
expenses2015			Yearly Expenses	8	num	expenses	2015	2
debt2015			Expenses>Income	8	char	debt	2015	3
income2016			Yearly Income	8	num	income	2016	1
expenses2016			Yearly Expenses	8	num	expenses	2016	2
debt2016			Expenses>Income	8	char	debt	2016	3
income2017			Yearly Income	8	num	income	2017	1
expenses2017			Yearly Expenses	8	num	expenses	2017	2
debt2017			Expenses>Income	8	char	debt	2017	3

How the Macro Works – Untranspose when there isn't an ID variable. The macro's final datastep has three sections: one to untranspose data when there isn't an *id* variable, another to accomplish the task when a long file is requested and, finally, a section to accomplish all other untransposition tasks. The datastep could have been written as a single section, but we decided that splitting the sections would make the code easier to both follow and modify. The case where there isn't an *id* variable is addressed first. This section of the datastep creates a long dataset where each record contains any *by* and *copy* variables, along with one *var* variable for each level of the *by* variable(s). When reading the first record from the T_E_M_P dataset (i.e., when _n_ eq 1), the macro uses CALL EXECUTE to write a data step that will produce the desired output file. The lengths of all output variables are declared early in the datastep in order to enhance processing speed:

```
data _null_;
  length forexec $255;
  set t_e_m_p end=lastone;
  by id_value;
  %if %length(&id) lt 1 %then %do;
    if _n_ eq 1 then do;
      call execute("data &libname_out..&out.");
      call execute("(&odsoptions. keep=&by. _name_ _value_ &copy.);");
      %if %length(%unquote(&dsoptions.)) gt 2 %then
         call execute("set &libname_in..&data. (&dsoptions.);");;
      %else call execute("set &libname_in..&data.;");;
      forexec="length _name_ $32 _value_ ";
      %if %length(&max_length) gt 0 %then %do;
        %if &mintype. eq char %then
          forexec= catt(forexec,"$",&max_length.,";");;
        %else forexec=catt(forexec,&max_length.,";");;
      %end;
      %else %do;
        %if &mintype. eq char %then
           forexec= catt(forexec,"$",&maxlength.,";");;
        %else forexec=catt(forexec,&maxlength.,";");;
      %end;
      call execute(forexec);
    end;
```

As you can see, the T_E_M_P dataset (Example 3), together with the macro variables created earlier in the code, provide all of the information necessary to accomplish the untransposition. The next part of this section, as shown below, is run for each record in the T_E_M_P dataset. Basically, it simply writes the necessary code to extract the desired variable names and values and outputs both missing and non-missing values unless the user included the *missing* parameter. The lines submitted to CALL EXECUTE are sent to a buffer and won't be executed until after the completion of the data step:

```
    forexec=catt('_name_="',var,'";');
    call execute(forexec);
    if type eq 'num' and "&mintype." eq "char" then
      forexec=catt('_value_=left(put(',name,',8.));');
    else forexec=catt('_value_=',name,';');
    call execute(forexec);
    %if %upcase(&missing.) eq NO %then %do;
      forexec=catt('if not missing(',name,') then do;');
      call execute(forexec);
    %end;
    call execute('output;');
    %if %upcase(&missing.) eq NO %then %do;
      call execute('end;');
    %end;
```

How the Macro Works – Untranspose when a long file is requested. The second section of the final datastep was designed to create a long dataset where each record contains any *by*, *id*, and *copy* variables, along with one *var* variable for each level of the *by* variable(s) and *id* level combinations. This section of the code would be invoked when one both declares an *id* variable and includes the *makelong=yes* parameter. When reading the first record from the T_E_M_P dataset (i.e., when _n_ eq 1), the macro uses CALL EXECUTE to write a data step that will produce the desired output file:

```
%else %if %upcase(&makelong.) eq YES %then %do;
  if _n_ eq 1 then do;
    call execute("data &libname_out..&out.");
    call execute("(&odsoptions. keep=&by. &id. _name_ _value_ &copy.);");
    %if %length(%unquote(&dsoptions.)) gt 2 %then %do;
      call execute("set &libname_in..&data. (&dsoptions.);");
    %end;
    %else %do;
      call execute("set &libname_in..&data.;");
    %end;
    forexec=catx(' ','informat',"&id.","&id_informat.",';');
    call execute(forexec);
    forexec=catx(' ','format',"&id.","&id_format.",';');
    call execute(forexec);
    forexec="length _name_ $32 _value_ ";
    %if %length(&max_length) gt 0 %then %do;
      %if &mintype.=char %then forexec=catt(forexec,"$",&max_length.,";");;
      %else forexec=catt(forexec,&max_length.,";");;
    %end;
    %else %do;
      %if &mintype.=char %then forexec=catt(forexec,"$",&maxlength.,";");;
      %else forexec=catt(forexec,&maxlength.,";");;
    %end;
    call execute(forexec);
  end;
```

Again, the T_E_M_P dataset (Example 3), together with the macro variables created earlier in the code, provide all of the information necessary to accomplish the untransposition. The next part of this section, as shown below, is run for each record in the T_E_M_P dataset. Basically, it simply writes the necessary code to extract the desired variable names and values and outputs both missing and non-missing values unless the user includes the *missing* parameter:

```
    forexec=catt('_name_="',var,'";');
    call execute(forexec);
    if type eq 'num' and "&mintype." eq "char" then
      forexec=catt('_value_=left(put(',name,',8.));');
    else forexec=catt('_value_=',name,';');
    call execute(forexec);
    %if %upcase(&missing.) eq NO %then %do;
      forexec=catt('if not missing(',name,') then do;');
      call execute(forexec);
    %end;
    if first("&id_informat.") ne "$" then do;
      makeid=input(id_value,&id_informat.);
      forexec=catt("&id.",'=',makeid,';');
    end;
    else do;
      makeid=put(id_value,&id_informat.);
      forexec=catt("&id.",'="',makeid,'";');
    end;
```

```
    call execute(forexec);
    call execute('output;');
    %if %upcase(&missing.) eq NO %then %do;
      call execute('end;');
    %end;
  %end;
```

How the Macro Works – Making a wide file less wide. The last section of the datastep was designed to create less wide datasets where each record contains any *by*, *id*, and *copy* variables, along with all of the *var* variables for each level of the *by* variable(s) and *id* level combinations. This section of the code would be invoked when one declares an *id* variable, but doesn't include the *makelong=yes* parameter. Like with the previous sections, the macro uses CALL EXECUTE to write a data step that will produce the desired output file. When reading the first record from the T_E_M_P dataset (i.e., when _n_ eq 1), the macro uses CALL EXECUTE to write a data step that will produce the desired output file. Unlike the two earlier sections, variable lengths aren't declared in the initial section as they have to be declared, separately, for each variable. We'll show the code produced by this section as it would be the one run for our example:

```
  %else %do;
    if _n_ eq 1 then do;
      call execute("data &libname_out..&out.");
      call execute("(&odsoptions. keep=&by. &id. &var. &copy.);");
      %if %length(%unquote(&dsoptions.)) gt 2 %then %do;
         call execute("set &libname_in..&data. (&dsoptions.);");
      %end;
      %else %do;
        call execute("set &libname_in..&data.;");
      %end;
      forexec=catx(' ','informat',"&id.","&id_informat.",';');
      call execute(forexec);
      forexec=catx(' ','format',"&id.","&id_format.",';');
      call execute(forexec);
      counter=1;
    end;
```

The lines created by that section of code are shown below:

```
data work.want (keep=id,year,income, expenses,debt);
  set work.have;
  informat year 8.;
  format year 8.;
```

Again, the T_E_M_P dataset (Example 3), together with the macro variables created earlier in the code, provide all of the information necessary to accomplish the untransposition. The next part of this section, as shown below, is only run for each variable identified in the *var* parameter. The code accomplishes that by establishing a counter variable that only has a value of 1 for the lowest level of the *id* variable (i.e., when the variable id_value equals 2015 in the example). This section of the code creates the statements that will set the labels, informats, format and lengths for each of the variables that will be output:

```
  if counter eq 1 then do;
    if not missing(label) then do;
      forexec=catx(' ','label',var,'=',label,';');
      call execute(forexec);
    end;
    if not missing(informat) then do;
      forexec=catx(' ','informat',var,informat,';');
```

```
      call execute(forexec);
    end;
    if not missing(format) then do;
      forexec=catx(' ','format',var,format,';');
      call execute(forexec);
    end;
    if not missing(length) then do;
      if type eq 'char' then forexec=catx(' ','length',var,'$',length,';');
      else forexec=catx(' ','length',var,length,';');
      call execute(forexec);
    end;
  end;
```

Since none of the variables had formats or informats declared, the lines created by the above calls will be:

```
income=income2015;
label income='Household Income';
length income 8 ;
expenses=expenses2015;
label expenses='Household Expenses';
length expenses 8 ;
debt=debt2015;
label debt='Household Debt';
length debt $ 8 ;
```

Finally, the following code is run for each record in the T_E_M_P dataset. Basically, it simply writes the necessary code to extract the desired *id* values, variable names and variable values, and outputs both missing and non-missing values unless the user includes the *missing* parameter:

```
    forexec=catt(var,'=',name,';');
    call execute(forexec);
    if last.id_value then do;
      counter+1;
      %if %upcase(&missing.) eq NO %then  %do;
        forexec=catx(' ','if',"&check",'then do;');
        call execute(forexec);
      %end;
      if first("&id_informat.") ne "$" then do;
        makeid=input(id_value,&id_informat.);
        forexec=catt("&id.",'=',makeid,';');
      end;
      else do;
        makeid=put(id_value,&id_informat.);
        forexec=catt("&id.",'="',makeid,'";');
      end;
      call execute(forexec);
      call execute('output;');
      %if %upcase(&missing.) eq NO %then call execute('end;');;
    end;
  %end;
```

Since the *missing* parameter wasn't changed from its default value of *NO* the lines created by the above code will be like the ones shown on the following page:

```
if (not missing(income)) or (not missing(expenses)) or (not missing(debt))
then do;
    year=2015;
    output;
end;
income=income2016;
expenses=expenses2016;
debt=debt2016;
if (not missing(income)) or (not missing(expenses)) or (not missing(debt))
then do;
    year=2016;
    output;
end;
income=income2017;
expenses=expenses2017;
debt=debt2017;
if (not missing(income)) or (not missing(expenses)) or (not missing(debt))
then do;
    year=2017;
    output;
end;
```

How the Macro Works – Ending the Datastep and Causing the Call Executed Statements to Run. You may recall that when the T_E_M_P dataset was set in the third line of the datastep, it was done so with the following statement: set t_e_m_p end=lastone. As such, the variable *lastone* will only be true when the last record of the T_E_M_P dataset has been processed. Thus, when it is, the following line:

```
if lastone then call execute('run;');
```

will send the following statement to the CALL EXECUTE buffer:

```
run;
```

Finally, the datastep's *run;* statement ends the data step and causes the statements in the CALL EXECUTE buffer to be submitted.

How the Macro Works – Creating a File Containing the Metadata. If a one or two-level filename was specified in the *metadata* parameter, the next section of the code will create a file containing the untransposed variables' metadata:

```
%if %length(&metadata) gt 0 %then %do;
  proc sql noprint;
    create table &metadata. as
      select distinct var as _name_, format as _format_,
             informat as _informat_, label as _label_,
             length as _length_, type as _type_
        from t_e_m_p
          order by order
    ;
  quit;
%end;
```

We didn't include the metadata parameter in our example call to the macro since, in the example, variables' names, formats, informats, labels, lengths and types were automatically set for all of the variables. However, if one did want to have the file created anyway, they would only have to add the metadata parameter. For example, to create a file in their work directory called metadata, they would only have to change the call to be:

```
%untranspose(data=have,out=want,by=id,id=year,var=income expenses debt,
 metadata=metadata)
```

The file metadata would be written to the user's work library and would look like the one shown in Example 4, below.

Example 4

name	_format_	_informat_	_label_	_length_	_type_
income			Yearly Income	8	num
expenses			Yearly Expenses	8	num
debt			Expenses>Income	8	char

How the Macro Works – Cleaning Up the User's Workspace. Since only one temporary file was created during the entire macro, the following was all that was needed to delete it:

```
   proc delete data=work.t_e_m_p;
   run;
%mend untranspose;
```

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, Powerpoint presentation, and this paper. The page can be found at: http://www.sascommunity.org/wiki/An_Easier_and_Faster_Way_to_Untranspose_a_Wide_File

CONCLUSION

The purpose of the present paper was to create, describe and share a SAS macro that could be used to complete complex data untranspositions 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.

Indeed, the macro allows users to complete most data untranspositions faster than using any method that involves PROC TRANSPOSE, and it requires less user input, fewer system resources, and includes more capabilities than PROC TRANSPOSE. Further, we believe that the macro's learning curve will be far less steep than the one confronted by most in learning to be proficient with PROC TRANSPOSE.

REFERENCES

A Better Way to Flip (Transpose) a SAS® Dataset, Tabachneck, A., Ke Shan, X., Virgile, R and Whitehurst, J., SGF 2013,http://www.sascommunity.org/wiki/A_Better_Way_to_Flip_(Transpose)_a_SAS_Dataset

Transpose Data with Macro %MAKEWIDE and %MAKELONG, Svolba, G., sasCommunity.org, 2016, http://www.sascommunity.org/wiki/Transpose_data_with_macro_%25MAKEWIDE_and_%25MAKELONG_(based_on_Proc_TRANSPOSE)

CONTACT INFORMATION

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

Arthur Tabachneck, Ph.D., CEO
AnalystFinder.com
Thornhill, ON Canada
E-mail: art@analystfinder.com

Gerhard Svolba, Ph.D.
SAS Institute, Inc.
Wien, Austria
E-mail: Gerhard.Svolba@sas.com

Joe Matise
NORC at the University of Chicago
Chicago, IL
E-mail: snoopy369@gmail.com

Matt Kastin
NORC at the University of Chicago
Chicago, IL
E-mail: matthew.kastin@gmail.com

APPENDIX A

/*THE %UNTRANSPOSE MACRO*/

```
/** The %untranspose macro
  *
  * This macro untransposes wider SAS datasets back to either the less wide
  * state that existed before the file was transposed, or to a long file
  *
  * AUTHORS: Arthur Tabachneck, Gerhard Svolba, Joe Matise and Matt Kastin
  * CREATED: September 25, 2017
  * MODIFIED: April 3, 2018

  Parameter Descriptions:

  *libname_in* NOT REQUIRED - the parameter to which you can assign the name
   of the SAS library that contains the dataset you want to untranspose. 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* NOT REQUIRED - the parameter to which you can assign the name
   of the SAS library where you want the untransposed 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* REQUIRED - the parameter to which you would assign the name of the
   file that you want to untranspose.  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. Additionally, as with PROC TRANSPOSE,
   the data parameter will also accept data step options. Thus, for example,
   if you had a dataset called 'have' and want to limit the untransposition
   to just the first 10 records, you could specify it as:
   data=have (obs=10).
   Any data step options accepted by a SAS data step can be included

  *out* REQUIRED - the parameter to which you would assign the name of the
   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* ONLY NECESSARY IF YOU HAVE A BY VARIABLE - the parameter to which you
   would assign the name of the dataset's by variable(s). The by parameter is
   like the by statement used in PROC TRANSPOSE, namely the identification of
   the variable (if any) that had been used to form by groups. By groups
   define the record level of the wide file you want to untranspose

  *prefix* ONLY NECESSARY IF YOUR TRANSPOSED VARIABLE NAME(S) BEGIN WITH A
   PREFIX - This is the parameter to which you would assign the string (if
   any) that each transposed variable begins with

  *var* REQUIRED the parameter to which you would assign the name or names of
   the original variables that had been transposed
```

id ONLY NECESSARY IF YOUR TRANSPOSED VARIABLE NAMES CONTAIN ID VALUES - the parameter to which you specify the variable name that was used as the ID variable when the transposed file was created. Only one variable can be assigned

id_informat ONLY NECESSARY IF 8. SHOULD NOT BE USED AS THE INFORMAT FOR EXTRACTING ID VALUES - the parameter to which you would assign the informat you want used for extracting the id variable's values

id_format ONLY NECESSARY IF 8. SHOULD NOT BE ASSIGNED AS THE FORMAT FOR EXTRACTED ID VALUES - the parameter to which you would indicate the format you want assigned to the id variable

var_first ONLY NECESSARY IF ID VALUES PRECEDE VARIABLE NAMES IN THE TRANSPOSED VARIABLE NAMES or IF THE TRANSPOSED VARIABLE NAME(S) DON'T INCLUDE THE VARIABLE NAME - the parameter that defines whether var names precede id values in the transposed variable names. Possible values are YES, NO or N/A and must be correctly assigned to reflect the way the transposed variables were formed

YES=[prefix]var[delimiter]id[suffix]
NO=[prefix]id[delimiter]var[suffix]
N/A=[prefix]id[suffix] or [prefix]+var[suffix]

delimiter ONLY NECESSARY IF YOUR TRANSPOSED VARIABLE NAME(S) CONTAIN A DELIMITER - the parameter to which you would assign the string (if any) that was used to separate var and ID values

suffix ONLY NECESSARY IF YOUR TRANSPOSED VARIABLE NAME(S) END WITH A SUFFIX - the parameter to which you would assign a string (if any) that each transposed variable ends with

copy ONLY NECESSARY IF YOUR WIDE FILE CONTAINS ONE OR MORE VARIABLES THAT SHOULD BE COPIED RATHER THAN UNTRANSPOSED - the parameter to which you would assign the name(s) of any variables that had been copied rather than transposed

missing ONLY NECESSARY IF YOU WANT TO OUTPUT A RECORD EVEN IF THE ONLY NON-MISSING VARIABLES ARE BY, ID OR COPY VARIABLES - PROC TRANSPOSE will output untransposed records even if the only non-missing variables are the BY, ID and COPY variables. If you want the macro to behave similarly set this parameter to equal YES

metadata NOT REQUIRED - the parameter to which you would specify the one or two-level SAS dataset the you want created to reflect the variable names, labels, informats, formats and types of the untransposed variables

makelong ONLY NECESSARY IF YOU WANT TO OUTPUT A SEPARATE RECORD FOR EACH BY VARIABLE, ID VALUE AND VARIABLE COMBINATION - this parameter will automatically be set to YES if no ID variable is declared with the ID parameter. If you do declare an ID variable, set this parameter to YES if you want the macro to output a long file

max_length (ONLY NECESSARY IF YOU WANT TO CONTROL THE LENGTH OF ALL UNTRANSPOSED VARIABLES) - This parameter is only applicable to those cases

```
   where you are untransposing a file from being wide to being long. If used
   it should be used cautiously as it could result in losing data
*/

%macro untranspose(libname_in=,
                   libname_out=,
                   data=,
                   out=,
                   by=,
                   prefix=,
                   var=,
                   id=,
                   id_informat=8.,
                   id_format=8.,
                   var_first=yes,
                   delimiter=,
                   suffix=,
                   copy=,
                   missing=NO,
                   metadata=,
                   makelong=,
                   max_length=);

/*Check whether the data and out parameters contain 1 or 2-level filenames*/
/*and, if needed, separate libname and data from data set options */
  %let lp=%sysfunc(findc(%superq(data),%str(%()));
  %if &lp. %then %do;
   %let rp=%sysfunc(findc(%superq(data),%str(%)),b));
/*for SAS*/
   %let dsoptions=%qsysfunc(substrn(%nrstr(%superq(data)),&lp+1,&rp-&lp-1));
   %let data=%sysfunc(substrn(%nrstr(%superq(data)),1,%eval(&lp-1)));
/*for WPS
   %let dsoptions=%qsysfunc(substrn(%nrquote(%superq(data)),&lp+1,&rp-&lp-
1));
   %let data=%sysfunc(substrn(%nrquote(%superq(data)),1,%eval(&lp-1)));
*/
  %end;
  %else %let dsoptions=;

  %let lp=%sysfunc(findc(%superq(out),%str(%()));
  %if &lp. %then %do;
   %let rp=%sysfunc(findc(%superq(out),%str(%)),b));
/*for SAS*/
   %let odsoptions=%qsysfunc(substrn(%nrstr(%superq(out)),&lp+1,&rp-&lp-1));
   %let out=%sysfunc(substrn(%nrstr(%superq(out)),1,%eval(&lp-1)));
/*for WPS
   %let odsoptions=%qsysfunc(substrn(%nrquote(%superq(out)),&lp+1,&rp-&lp-
1));
   %let out=%sysfunc(substrn(%nrquote(%superq(out)),1,%eval(&lp-1)));
*/
  %end;
  %else %let odsoptions=;
  %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;

/*Create macro variable to contain a list of variables that were copied*/
  %let to_copy=;
  %if %length(&copy.) gt 0 %then %do;
    data t_e_m_p;
      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="T_E_M_P"
        ;
      quit;
  %end;

  data t_e_m_p;
    array vars(*) &var.;
    output;
  run;

  proc sql noprint;
    select catt("'",name,"'"),
           catt('(not missing(',name,'))')
       into :vars separated by ",",
            :check separated by " or "
          from dictionary.columns
            where libname="WORK" and
                  memname="T_E_M_P"
              order by length(name) descending
    ;

    select catt("'",name,"'")
       into :ordered_vars separated by ","
          from dictionary.columns
            where libname="WORK" and
                  memname="T_E_M_P"
              order by varnum
    ;
  quit;

  data t_e_m_p;
    set &libname_in..&data. (obs=1 &dsoptions.
    %if %length(&by.) gt 0 or %length(&copy) gt 0 %then drop=&by. &copy.;);
  run;
```

```
  proc sql noprint;
    create table t_e_m_p as
      select name,format,informat,label,length,type
         from dictionary.columns
           where libname="WORK" and
                 memname="T_E_M_P"
    ;
    select min(type), max(length)
      into :mintype,:maxlength
        from WORK.T_E_M_P
    ;
  quit;

  data t_e_m_p (drop=temp);
    set t_e_m_p;
    do var=&vars.;
      if %length(&id) gt 0 then do;
        if upcase("&var_first.") eq 'YES' then do;
          if catt(upcase("&prefix."),upcase(var))=:strip(upcase(name)) then
do;
            id_value=substr(strip(name),%length(&prefix.)+length(strip(var))+
                %length(&delimiter)+1,
                length(strip(name))-%length(&prefix.)-length(strip(var))-
                %length(&delimiter)-%length(&suffix.));
            leave;
          end;
        end;
        else if upcase("&var_first.") eq 'N/A' then do;
          id_value=substr(strip(name),%length(&prefix.)+1,
                length(strip(name))-%length(&prefix.)-%length(&suffix.));
        end;
        else do;
          if strip(reverse(catt(upcase(var),upcase("&suffix.")))) =:
             strip(reverse(upcase(name))) then do;
            temp=reverse(substr(reverse(strip(name)),
                %length(&suffix)+length(strip(var))+%length(&delimiter)+1));
            id_value=substr(strip(temp),%length(&prefix.)+1);
            leave;
          end;
        end;
      end;
      else do;
        if catt(upcase("&prefix."),upcase(var),upcase("&suffix."))=:
            strip(upcase(name)) then do;
          id_value='1';
          leave;
        end;
      end;
    end;
    order=0;
    do temp=&ordered_vars.;
      order+1;
      if strip(upcase(var)) eq strip(upcase(temp)) then leave;
    end;
  run;
```

```
proc sort data=t_e_m_p;
  by id_value order;
run;

data _null_;
  length forexec $255;
  set t_e_m_p end=lastone;
  by id_value;
  %if %length(&id) lt 1 %then %do;
    if _n_ eq 1 then do;
      call execute("data &libname_out..&out.");
      call execute("(&odsoptions. keep=&by. _name_ _value_ &copy.);");
      %if %length(%unquote(&dsoptions.)) gt 2 %then %do;
         call execute("set &libname_in..&data. (&dsoptions.);");
      %end;
      %else %do;
        call execute("set &libname_in..&data.;");
      %end;
      forexec="length _name_ $32 _value_ ";
      %if %length(&max_length) gt 0 %then %do;
        %if &mintype. eq char %then %do;
          forexec=catt(forexec,"$",&max_length.,";");
        %end;
        %else %do;
          forexec=catx(' ',forexec,&max_length.,";");
        %end;
      %end;
      %else %do;
        %if &mintype. eq char %then %do;
          forexec=catt(forexec,"$",&maxlength.,";");
        %end;
        %else %do;
          forexec=catx(' ',forexec,&maxlength.,";");
        %end;
      %end;
      call execute(forexec);
    end;
    forexec=catt('_name_="',var,'";');
    call execute(forexec);
    if type eq 'num' and "&mintype." eq "char" then
      forexec=catt('_value_=left(put(',name,',8.));');
    else forexec=catt('_value_=',name,';');
    call execute(forexec);
    %if %upcase(&missing.) eq NO %then %do;
      forexec=catt('if not missing(',name,') then do;');
      call execute(forexec);
    %end;
    call execute('output;');
    %if %upcase(&missing.) eq NO %then %do;
      call execute('end;');
    %end;
  %end;
  %else %if %upcase(&makelong.) eq YES %then %do;
    if _n_ eq 1 then do;
      call execute("data &libname_out..&out.");
      call execute("(&odsoptions. keep=&by. &id. _name_ _value_ &copy.);");
      %if %length(%unquote(&dsoptions.)) gt 2 %then %do;
```

```
       call execute("set &libname_in..&data. (&dsoptions.);");
    %end;
    %else %do;
      call execute("set &libname_in..&data.;");
    %end;
    forexec=catx(' ','informat',"&id.","&id_informat.",';');
    call execute(forexec);
    forexec=catx(' ','format',"&id.","&id_format.",';');
    call execute(forexec);
    forexec="length _name_ $32 _value_ ";
    %if %length(&max_length) gt 0 %then %do;
      %if &mintype. eq char %then %do;
        forexec=catt(forexec,"$",&max_length.,";");
      %end;
      %else %do;
        forexec=catx(' ',forexec,&max_length.,";");
      %end;
    %end;
    %else %do;
      %if &mintype. eq char %then %do;
        forexec=catt(forexec,"$",&maxlength.,";");
      %end;
      %else %do;
        forexec=catx(' ',forexec,&maxlength.,";");
      %end;
    %end;
    call execute(forexec);
  end;
  forexec=catt('_name_="',var,'";');
  call execute(forexec);
  if type eq 'num' and "&mintype." eq "char" then
    forexec=catt('_value_=left(put(',name,',8.));');
  else forexec=catt('_value_=',name,';');
  call execute(forexec);
  %if %upcase(&missing.) eq NO %then %do;
    forexec=catt('if not missing(',name,') then do;');
    call execute(forexec);
  %end;
  if first("&id_informat.") ne "$" then do;
    makeid=input(id_value,&id_informat.);
    forexec=catt("&id.",'=',makeid,';');
  end;
  else do;
    makeid=put(id_value,&id_informat.);
    forexec=catt("&id.",'="',makeid,'";');
  end;
  call execute(forexec);
  call execute('output;');
  %if %upcase(&missing.) eq NO %then %do;
    call execute('end;');
  %end;
%end;
%else %do;
  if _n_ eq 1 then do;
    call execute("data &libname_out..&out.");
    call execute("(&odsoptions. keep=&by. &id. &var. &copy.);");
    %if %length(%unquote(&dsoptions.)) gt 2 %then %do;
```

```
           call execute("set &libname_in..&data. (&dsoptions.);");
        %end;
        %else %do;
          call execute("set &libname_in..&data.;");
        %end;
        forexec=catx(' ','informat',"&id.","&id_informat.",';');
        call execute(forexec);
        forexec=catx(' ','format',"&id.","&id_format.",';');
        call execute(forexec);
        counter=1;
      end;
      if counter eq 1 then do;
        if not missing(label) then do;
          forexec=catx(' ','label',var,'=',label,';');
          call execute(forexec);
        end;
        if not missing(informat) then do;
          forexec=catx(' ','informat',var,informat,';');
          call execute(forexec);
        end;
        if not missing(format) then do;
          forexec=catx(' ','format',var,format,';');
          call execute(forexec);
        end;
        if not missing(length) then do;
          if type eq 'char' then forexec=catx('
','length',var,'$',length,';');
          else forexec=catx(' ','length',var,length,';');
          call execute(forexec);
        end;
      end;
      forexec=catt(var,'=',name,';');
      call execute(forexec);
      if last.id_value then do;
        counter+1;
        %if %upcase(&missing.) eq NO %then  %do;
          forexec=catx(' ','if',"&check",'then do;');
          call execute(forexec);
        %end;
        if first("&id_informat.") ne "$" then do;
          makeid=input(id_value,&id_informat.);
          forexec=catt("&id.",'=',makeid,';');
        end;
        else do;
          makeid=put(id_value,&id_informat.);
          forexec=catt("&id.",'="',makeid,'";');
        end;
        call execute(forexec);
        call execute('output;');
        %if %upcase(&missing.) eq NO %then call execute('end;');;
      end;
    %end;
    if lastone then call execute('run;');
  run;

  %if %length(&metadata) gt 0 %then %do;
    proc sql noprint;
```

```
      create table &metadata. as
        select distinct var as _name_, format as _format_,
               informat as _informat_, label as _label_,
               length as _length_, type as _type_
          from t_e_m_p
            order by order
      ;
    quit;
  %end;

/*Delete all temporary files*/
   proc delete data=work.t_e_m_p;
   run;
%mend untranspose;

/****************Examples*********************
*3 variables with the variable names formatted as var+id;
data have;
  input id income2015-income2017
           expenses2015-expenses2017
           (debt2015-debt2017) ($);
  cards;
1 70000 75500 80000 60000 70000 81000 no no yes
2 50000 52000 55000 42000 53000 60000 no yes yes
3 80000 90000 99000 70000 75000 85000 no  no  no
;

%untranspose(data=have, out=want, by=id, id=year,
 var=income expenses debt)

*3 variables with the variable names formatted as var+id, but only
untransposing the first obs;

%untranspose(data=have (obs=1), out=want, by=id, id=year,
 var=income expenses debt)

*3 variables with the variable names formatted as var+id, but only
 untransposing a specific id;

%untranspose(data=have (where=(id eq 2)), out=want, by=id, id=year,
 var=income expenses debt)

*3 variables with the variable names formatted as var+delimiter+id;
data have;
  input id income_2015-income_2017
           expenses_2015-expenses_2017
           (debt_2015-debt_2017) ($);
  cards;
1 70000 75500 80000 60000 70000 81000 no no yes
2 50000 52000 55000 42000 53000 60000 no yes yes
3 80000 90000 99000 70000 75000 85000 no  no  no
;

%untranspose(data=have, out=want, by=id, delimiter=_, id=year,
 var=income expenses debt)

*3 variables with the variable names formatted as var+delimiter+id, but
```

```
 changing the order of the variables output;
data have;
  input id income_2015-income_2017
           expenses_2015-expenses_2017
           (debt_2015-debt_2017) ($);
  cards;
1 70000 75500 80000 60000 70000 81000 no no yes
2 50000 52000 55000 42000 53000 60000 no yes yes
3 80000 90000 99000 70000 75000 85000 no  no  no
;
%untranspose(data=have, out=want, by=id, delimiter=_, id=year,
 var=debt income expenses)

*1 variable with the variable names formatted as: var+id;
data have;
  input weight1-weight3;
  cards;
77 79 83
;
%untranspose(data=have, out=want, id=time, var=weight)

*1 variable with the variable names formatted as:
 prefix+var+delimiter+id+suffix;
data have;
  input id _this_1_test _this_2_test _this_3_test;
  cards;
1 1 2 3
2 6 5 4
;
%untranspose(data=have, out=want, by=id, prefix=_, id=qtr, delimiter=_,
 var=this,suffix=_test)

*2 variables with the variable names formatted as:
 prefix+var+delimiter+id+suffix;
data have;
  input id _this_1_test _this_2_test _this_3_test
        _thiss_1_test _thiss_2_test _thiss_3_test;
  cards;
1 1 2 3 4 5 6
2 6 5 4 3 2 1
;

%untranspose(data=have, out=want, by=id, prefix=_, id=qtr, delimiter=_,
 var=this thiss,suffix=_test)

*2 variables with the variable names formatted as:
 prefix+id+delimiter+var+suffix;
data have;
  input id _1_this_test _2_this_test _3_this_test _1_thiss_test _2_thiss_test
_3_thiss_test;
  cards;
1 1 2 3 4 5 6
2 6 5 4 3 2 1
;

%untranspose(data=have, out=want, by=id, prefix=_, id=qtr, delimiter=_,
 var_first=no,var=this thiss, suffix=_test)
```

```
*2 variables with the variable names formatted as:
 prefix+id+delimiter+var+suffix;
data have;
  input id thisA thisB thisC
        (thisislongerA thisislongerB thisislongerC) ($);
  label thisA='Shorter';
  label thisB='Shorter';
  label thisC='Shorter';
  label thisislongerA='Longer';
  label thisislongerB='Longer';
  label thisislongerC='Longer';
  cards;
1 1 2 3 D E F
2 6 5 4 C B A
;

%untranspose(data=have, out=want, by=id, id=section,
 var=this thisislonger, id_informat=$1.,id_format=$1.)

*or to only transpose one or some of the variables:

%untranspose(data=have(keep=id thisA--thisC), out=want, by=id, id=section,
 var=this, id_informat=$1.,id_format=$1.)

*1 variable with the variable names formatted as: prefix+id;
data have;
  informat customer 8.
    _0-_6 $12.;
  input customer (_0-_6) (&);
  cards;
1 herring  corned beef  olives  ham  turkey  bourbon  ice cream
2 corned beef  peppers  bourbon  crackers  chicken  ice cream  ice cream
;

%untranspose(data=have, out=want, id=time,prefix=_,var_first=n/a,
 var=product, id_informat=1.0,id_format=1.0,by=customer)

*6 variables with three formatted and the variable names formatted as: var;
proc format;
  value n
  1='AA'
  2='BB'
  3='CC'
  ;
  value $c
  'A'='11'
  'B'='22'
  ;
run;

data have;
  length subject 8;
  label var1='first var'
        var2='second var'
        var3='third var'
        var4='fourth var'
```

```
        var5='fifth var'
        var6='sixth var'
  ;
  format var2 n.
         var3 comma6.
         var4 $c.;

  input subject var1-var3 (var4-var6) ($);
  cards;
1 1 2 30000 A B this
2 3 2 10000 B A that
;

%untranspose(data=have, out=want, var=var1-var6, by=subject, metadata=meta,
max_length=5)

*6 variables with three formatted and the variable names formatted as:
prefix+var;
proc format;
  value n
  1='AA'
  2='BB'
  3='CC'
  ;
  value $c
  'A'='11'
  'B'='22'
  ;
run;

data have;
  length subject 8;
  label test_var1='first var'
        test_var2='second var'
        test_var3='third var'
        test_var4='fourth var'
        test_var5='fifth var'
        test_var6='sixth var'
  ;
  format test_var2 n.
         test_var3 comma6.
         test_var4 $c.;

  input subject test_var1-test_var3 (test_var4-test_var6) ($);
  cards;
1 1 2 30000 A B this
2 3 2 . B A that
;

%untranspose(data=have, out=want, var=var1-var6, by=subject, prefix=test_,
missing=yes, metadata=meta, max_length=5)
**************************************************************/
```

APPENDIX B

/*THE %UNTRANSPOSE MACRO TIP SHEET*/

Untranspose Macro Tip Sheet

Purpose: The macro untransposes wider SAS datasets back to either the less wide state that existed before the file was transposed, or to a long file. The macro can accommodate any prefixes, variable names, delimiters, ID values, and suffixes that may exist in the transposed variable names.

Named Parameters: The macro uses named parameters so that (1) default values can be assigned and (2) the various parameters 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.

Parameter	Required	Possible Values	Default Value	Description
libname_in	No	Any valid libname	work	The parameter to which you can assign the name of the SAS library that contains the dataset you want to untranspose
libname_out	No	Any valid libname	work	The parameter to which you can assign the name of the SAS library where you want the untransposed file written
data	Yes	Any valid filename	None	The parameter to which you assign the one or two-level name of the file that you want to untranspose
out	Yes	Any valid filename	None	The parameter to which you assign the name of the file that you want the macro to create
by	No	Any variable name from the file specified in the data parameter	None	The parameter to which you would assign the name of the original dataset's by variable(s)
prefix	No	Any valid SAS name characters	None	The parameter to which you assign the string (if any) that the transposed variable names begin with
var	Yes	Any valid SAS name	None	The parameter to which you assign the name(s) of the original variables that had been transposed
id	No	Any valid SAS name	None	The parameter to which you specify the variable name that was used as the ID variable (if any) when the transposed file was created. Only one variable can be assigned
Id_informat	No	Any valid SAS informat	8.	The parameter to which you can assign the informat to be used to extract the id variable's values
Id_format	No	Any valid SAS format	8.	The parameter to which you can indicate the format you want assigned to the id variable
var_first	No	YES=<prefix>var<delimiter>id<suffix> NO=<prefix>id<delimiter>var<suffix> N/A=<prefix>var<suffix>	Yes	The parameter that defines whether var names precede id values in the transposed variable names
delimiter	No	Any valid SAS name characters	None	The parameter to which you assign the string (if any) that was used to separate var and ID values in the transposed variable names
suffix	No	Any valid SAS name characters	None	The parameter to which you can assign a string (if any) that the transposed variable names end with
copy	No	Any valid SAS name	None	The parameter to which you can assign the name(s) of any variables that had been copied
missing	No	Yes or no (case insensitive)	No	The parameter to indicate whether a record should be output if the only non-missing variables are the BY, ID and COPY variables
metadata	No	Any valid filename	None	The parameter to which you can specify the one or two-level SAS dataset the you want created to contain the untransposed variables' metdata
makelong	No	Yes or no (case insensitive)	No	The parameter to which you can specify that you want the macro to output records at the BY variable, ID variable value, var variable(s) level
max_length	No	Any number between 1 and 32767	None	the parameter to which you can specify the length of the _value_ variable

Usage Examples: The following are some examples of how you might use the macro. For each example the wide dataset's name is *have* and resides in the work library, and the less wide or long dataset created by the macro is called *want* and also resides in the work library.

dataset: ***have***

id	income2015	income2016	income2017	expenses2015	expenses2016	expenses2017	debt2015	debt2016	debt2017
1	70000	75500	80000	60000	70000	81000	no	no	yes
2	50000	52000	55000	42000	53000	60000	no	yes	yes
3	80000	90000	99000	70000	75000	85000	no	no	no

macro call: ***%untranspose*(data=have, out=want, by=id, id=year, var=income expenses debt)**

dataset: ***want***

id	year	income	expenses	debt
1	2015	70000	60000	no
1	2016	75500	70000	no
1	2017	80000	81000	yes
2	2015	50000	42000	no
2	2016	52000	53000	yes
2	2017	55000	60000	yes
3	2015	80000	70000	no
3	2016	90000	75000	no
3	2017	99000	85000	no

dataset: ***have***

id	income2015	income2016	income2017	expenses2015	expenses2016	expenses2017	debt2015	debt2016	debt2017
1	70000	75500	80000	60000	70000	81000	no	no	yes
2	50000	52000	55000	42000	53000	60000	no	yes	yes
3	80000	90000	99000	70000	75000	85000	no	no	no

macro call: ***%untranspose*(data=have (obs=1), out=want, by=id, id=year, var=income expenses debt)**

dataset: ***want***

id	year	income	expenses	debt
1	2015	70000	60000	no
1	2016	75500	70000	no
1	2017	80000	81000	yes

dataset: ***have***

id	income2015	income2016	income2017	expenses2015	expenses2016	expenses2017	debt2015	debt2016	debt2017
1	70000	75500	80000	60000	70000	81000	no	no	yes
2	50000	52000	55000	42000	53000	60000	no	yes	yes
3	80000	90000	99000	70000	75000	85000	no	no	no

macro call: ***%untranspose*(data=have (where=(id eq 2)), out=want, by=id, id=year, var=income expenses debt)**

dataset: ***want***

id	year	income	expenses	debt
2	2015	50000	42000	no
2	2016	52000	53000	yes
2	2017	55000	60000	yes

dataset: ***have***

id	income_2015	income_2016	income_2017	expenses_2015	expenses_2016	expenses_2017	debt_2015	debt_2016	debt_2017
1	70000	75500	80000	60000	70000	81000	no	no	yes
2	50000	52000	55000	42000	53000	60000	no	yes	yes
3	80000	90000	99000	70000	75000	85000	no	no	no

macro call: ***%untranspose(data=have, out=want, by=id, delimiter=_, id=year, var=income expenses debt)***

dataset: ***want***

id	year	income	expenses	debt
1	2015	70000	60000	no
1	2016	75500	70000	no
1	2017	80000	81000	yes
2	2015	50000	42000	no
2	2016	52000	53000	yes
2	2017	55000	60000	yes
3	2015	80000	70000	no
3	2016	90000	75000	no
3	2017	99000	85000	no

dataset: ***have***

id	income_2015	income_2016	income_2017	expenses_2015	expenses_2016	expenses_2017	debt_2015	debt_2016	debt_2017
1	70000	75500	80000	60000	70000	81000	no	no	yes
2	50000	52000	55000	42000	53000	60000	no	yes	yes
3	80000	90000	99000	70000	75000	85000	no	no	no

macro call: ***%untranspose(data=have, out=want, by=id, delimiter=_, id=year, var=debt income expenses)***

dataset: ***want***

id	year	debt	income	expenses
1	2015	no	70000	60000
1	2016	no	75500	70000
1	2017	yes	80000	81000
2	2015	no	50000	42000
2	2016	yes	52000	53000
2	2017	yes	55000	60000
3	2015	no	80000	70000
3	2016	no	90000	75000
3	2017	no	99000	85000

dataset: ***have***

weight1	weight2	weight3
77	79	83

macro call: ***%untranspose(data=have, out=want, id=time, var=weight)***

dataset: ***want***

time	weight
1	77
2	79
3	83

dataset: ***have***

id	_this_1_test	_this_2_test	_this_3_test
1	1	2	3
2	6	5	4

macro call: %untranspose(data=have, out=want, by=id, prefix=_, id=qtr, delimiter=_, var=this, suffix=_test)

dataset: ***want***

id	qtr	this
1	1	1
1	2	2
1	3	3
2	1	6
2	2	5
2	3	4

dataset: ***have***

id	_this_1_test	_this_2_test	_this_3_test	_thiss_1_test	_thiss_2_test	_thiss_3_test
1	1	2	3	4	5	6
2	6	5	4	3	2	1

macro call: %untranspose(data=have, out=want, by=id, prefix=_, id=qtr, delimiter=_, var=this thiss, suffix=_test)

dataset: ***want***

id	qtr	this	thiss
1	1	1	4
1	2	2	5
1	3	3	6
2	1	6	3
2	2	5	2
2	3	4	1

dataset: ***have***

id	_1_this_test	_2_this_test	_3_this_test	_1_thiss_test	_2_thiss_test	_3_thiss_test
1	1	2	3	4	5	6
2	6	5	4	3	2	1

macro call: %untranspose(data=have, out=want, by=id, prefix=_, id=qtr, delimiter=_, var_first=no,var=this thiss, suffix=_test)

dataset: ***want***

id	qtr	this	thiss
1	1	1	4
1	2	2	5
1	3	3	6
2	1	6	3
2	2	5	2
2	3	4	1

dataset: ***have***

id	thisA	thisB	thisC	thisislongerA	thisislongerB	thisislongerC
1	1	2	3	D	E	F
2	6	5	4	C	B	A

macro call: **%untranspose(data=have, out=want, by=id, id=section, var=this thisislonger, id_informat=$1.,id_format=$1.)**

dataset: ***want***

id	section	this	thisislonger
1	A	1	D
1	B	2	E
1	C	3	F
2	A	6	C
2	B	5	B
2	C	4	A

dataset: ***have***

id	thisA	thisB	thisC	thisislongerA	thisislongerB	thisislongerC
1	1	2	3	D	E	F
2	6	5	4	C	B	A

macro call: **%untranspose(data=have(keep=id thisA--thisC), out=want, by=id, id=section, var=this, id_informat=$1.,id_format=$1.)**

dataset: ***want***

id	section	this
1	A	1
1	B	2
1	C	3
2	A	6
2	B	5
2	C	4

dataset: ***have***

customer	_0	_1	_2	_3	_4	_5	_6
1	herring	corned beef	olives	ham	turkey	bourbon	ice cream
2	corned beef	peppers	bourbon	crackers	chicken	ice cream	ice cream

macro call: **%untranspose(data=have, out=want, id=time, prefix=_, var_first=n/a,**
var=product, id_informat=1.0,id_format=1.0,by=customer)

dataset: ***want***

customer	time	product
1	0	herring
1	1	corned beef
1	2	olives
1	3	ham
1	4	turkey
1	5	bourbon
1	6	ice cream
2	0	corned beef
2	1	peppers
2	2	bourbon
2	3	crackers
2	4	chicken
2	5	ice cream
2	6	ice cream

dataset: ***have***

```
proc format;
  value n
  1='AA'
  2='BB'
  3='CC'
  ;
  value $c
  'A'='11'
  'B'='22'
  ;
run;

data have;
  length subject 8;
  label var1='first var'
        var2='second var'
        var3='third var'
        var4='fourth var'
        var5='fifth var'
        var6='sixth var'
  ;
  format var2 n.
         var3 comma6.
         var4 $c.;

  input subject var1-var3 (var4-var6) ($);
  cards;
1 1 2 30000 A B this
2 3 2 10000 B A that
;
```

Obs	subject	var1	var2	var3	var4	var5	var6
1	1	1	BB	30,000	11	B	this
2	2	3	BB	10,000	22	A	that

macro call:**%untranspose(data=have, out=want, var=var1-var6, by=subject, metadata=meta)**

dataset: ***want***

Obs	subject	_name_	_value_
1	1	var1	1
2	1	var2	2
3	1	var3	30000
4	1	var4	A
5	1	var5	B
6	1	var6	this
7	2	var1	3
8	2	var2	2
9	2	var3	10000
10	2	var4	B
11	2	var5	A
12	2	var6	that

dataset: **meta**

Obs	_name_	_format	_informat_	_label_	_length_	_type_
1	var1			first var	8	num
2	var2	N.		second var	8	num
3	var3	COMMA6.		third var	8	num
4	var4	$C.		fourth var	2	char
5	var5			fifth var	8	char
6	var6			sixth var	8	char