Paper TF12-2012

Atypical Applications of Proc Transpose

John King, Ouachita Clinical Data Services Inc., Hopper, AR

ABSTRACT

Did you know that PROC TRANSPOSE can be used to verify the existence of variables in a SAS® data set? It can process a list of variable name(s) or a "SAS Variable List" or a combination of the two. You can also check if the variables in a list are numeric or character. Using the return code from PROC TRANSPOSE a program can branch based on success or failure of PROC TRANSPOSE. Did you know you can use PROC TRANSPOSE and a data step to determine if a list of variables is in one variable list but not another? This paper shows how to accomplish these tasks and other atypical uses of PROC TRANSPOSE.

INTRODUCTION

Most often PROC TRANSPOSE is used to transpose the data contained in a SAS data set, in order to create more rows of data from variables or to create variables from observations. Creating variables from observations using metadata to derive the variable name is a unique feature of PROC TRANSPOSE that is not available in the data step. PROC TRANSPOSE can also create an extremely useful data set when it is asked to transpose variables when there are zero observations to transpose. The TRANSPOSE of zero observation produces a data with one observation for each variable listed in the VAR statement or for all numeric variables if no VAR statement is used.

CREATE A DATA SET OF ALL VARIABLE NAMES AND LABELS.

```
proc transpose
      data=sashelp.heart(obs=0)
      out=heartVars;
   var _all_;
   run;
```

When a data set with zero observations is transposed PROC TRANSPOSE simply creates a data set of variable names and labels. Most usually a VAR statement is included to specify which variables are transposed and the order in the output dataset. This technique will be modified slightly in the next few examples and hopefully it will become clear how useful this simple PROC step can be.

Obs	_NAME_	_LABEL_
1	Status	
2	DeathCause	Cause of Death
3	AgeCHDdiag	Age CHD Diagnosed
4-14	Omitted	
15	BP_Status	Blood Pressure Status
16	Weight_Status	Weight Status
17	Smoking_Status	Smoking Status

PROC TRANSPOSE has created data HEARTVARS with one observation for each variable name listed in the VAR statement. If a variable name is listed more than once there will be an observation for each time the variable is listed. The order of the variables in the data reflects the order of the variable in the VAR statement. The data will have at least one variable named by the PROC TRANSPOSE NAME= option. The default name is _NAME_. If at least one of the

variables also has a label the variable named by the PROC TRANSPOSE LABEL= option. The default label is _LABEL_.

```
data heartVars(index=(_name_ _order_));
   set heartVars;
   _ORDER_ + 1;
   run;
```

We can increase the usefulness of this by making it into and index lookup table by adding an index variable and indexing the table on _NAME_ and the new index variable.

Obs	_NAME_	_LABEL_	_ORDER_
1	Status		1
2	DeathCause	Cause of Death	2
3	AgeCHDdiag	Age CHD Diagnosed	3
4	Sex		4
5	AgeAtStart	Age at Start	5
6	Height		6
7	Weight		7
8	Diastolic		8
9	Systolic		9
10	MRW	Metropolitan Relative Weight	10
11	Smoking		11
12	AgeAtDeath	Age at Death	12
13	Cholesterol		13
14	Chol_Status	Cholesterol Status	14
15	BP_Status	Blood Pressure Status	15
16	Weight_Status	Weight Status	16
17	Smoking_Status	Smoking Status	17

With the addition variable _ORDER_ and indexing the data on both _NAME_ and _ORDER_ data HEARTVARS becomes a lookup table that can be used to lookup a variable's position in the variable list or to lookup a variable name base on position in the list. I have used this technique in programs like that described in Summary Statistics in Rows. Summary statistics in rows generates summaries of continuous and/or categorical variables in a stacked table with treatment as column a grouping, the typical demography table in most clinical trials. In this application the variable names passed to the program to be summarized later become values of a grouping variable. The program uses the indexed data set as a lookup table to lookup each variable's LABEL to create a row label variable in the table. The program also uses the order variable to determine the final sort order for the variables in the summary table.

DO VARIABLES EXIST?

```
%macro main(data=,var=);
   Proc transpose
         data=&data(obs=0)
         out=&sysmacroname._varlist;
      var &var;
      run;
   %put NOTE: SYSERR=&SYSERR;
   %if &syserr gt 0 %then %do;
      %put %str(ERR)OR: Macro &sysmacroname
ending due to errors;
      %return;
      %end;
   proc print data=&sysmacroname._varlist;
      run;
   %mend main;
```

Imagine a program that accepts a variable list and data set name, parameters DATA= and VAR=, as input and it is desired to expand the variable list and check that all names exist in the input data. If a variable named in the list is not found PROC TRANSPOSE will produce an ERROR: and set the automatic variable SYSERR to a value greater than 0. The program can than then test SYSERR in this case using a macro %IF statement and take an action deemed appropriate. In this example an error message is printed and the macro executes the %RETURN statement ending macro execution.

```
144  %main(DATA=sashelp.heart,var=AGE: SEX BMI);
ERROR: Variable BMI not found.

NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.MAIN_VARLIST may be incomplete.  When this step was stopped there were
0 observations and 0 variables.
WARNING: Data set WORK.MAIN_VARLIST was not replaced because this step was stopped.

NOTE: SYSERR=3000
ERROR: Macro MAIN ending due to errors
```

Also note that PROC TRANSPOSE prints an error message to indicate which variable(s) are not found. When the step is successful a data set of variable names is created. "SAS Variable Lists" e.g. "AGE:" are expanded and the variables are ordered in the same order they are specified on the VAR statement. "SAS Variable List" expansion is and import feature of the technique, letting SAS handle the details makes it easy.

Obs	_NAME_	_LABEL_	_ORDER_
1	AgeCHDdiag	Age CHD Diagnosed	1
2	AgeAtStart	Age at Start	2
3	AgeAtDeath	Age at Death	3
4	Sex		4
5	MRW	Metropolitan Relative Weight	5

When the TRANSPOSE is successful and the data VARLIST is created a data can be used to add an indexing variable _ORDER_ and the data can be index by _NAME_ and _ORDER_ transforming the data set into a very useful data object, an indexed lookup table.

IS THE LIST OF VARIABLES A SPECIFIC DATA TYPE?

```
%macro main(data=,var=);
   Proc transpose
         data=&data(obs=0 keep=_NUMERIC_)
         out=&sysmacroname._varlist;
      var &var;
      run;
   %put NOTE: SYSERR=&SYSERR;
   %if &syserr gt 0 %then %do;
      %put %str(ERR)OR: Variables listed as not
found are not numeric or do not exist.;
      %put %str(ERR)OR: Macro &sysmacroname
ending due to errors;
      %return;
      %end;
   proc sql noprint;
      select _NAME_ into :var separated by ' '
         from &sysmacroname._varlist;
      quit;
      run;
   %put NOTE: VAR=&var;
   proc summary data=&data;
      var &var;
      output out=&sysmacroname._summary;
      run;
   proc print;
      run;
   %mend main;
```

Expanding the test for existence we can use a similar technique to test that lists of variables not only exist but are of a specific type. Consider a macro designed to accept input where a list of variables is to be process and the data type must be numeric. Here a KEEP=_NUMERIC_ data set option is added to limit the variables processed to numeric variables only. Now the test for existence also includes a test for type. If a character variable is included in the list PROC TRANSPOSE ends with an error indicating the variable does not exist, because it was not kept. The test is now a test for both existence and type.

```
789  %main(DATA=sashelp.heart,var=AGE: SEX MRW);

MPRINT(MAIN):   Proc transpose data=sashelp.heart(obs=0 keep=_NUMERIC_) out=MAIN_varlist;
ERROR: Variable SEX not found.
MPRINT(MAIN):   var AGE: SEX MRW;
MPRINT(MAIN):   run;

NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.MAIN_VARLIST may be incomplete.  When this step was stopped there were
0 observations and 0 variables.
WARNING: Data set WORK.MAIN_VARLIST was not replaced because this step was stopped.

NOTE: SYSERR=3000
ERROR: Variables listed as not found are not numeric or do not exist.
ERROR: Macro MAIN ending due to errors.
```

Above the SAS log reveals that the variable SEX is not found in SASHELP.HEART when only numeric variables are included for processing. The macro prints a message and stops execution.

EXPAND A VARIABLE LIST INTO A DELIMITED LIST.

It is often helpful to expand a variable list, especially when it contains "SAS Variable Lists", into a list of individually specified variables. This would be useful in a macro that accepts "SAS Variable Lists" and to insure correct processing later on when the list is expanded into individual variable names. Another reason might be to normalize the variable names with respect to case, e.g. when a variable specified as "SEX" is actually "Sex" in the data set and it is desired to retrieve the "correct" spelling. There are also SAS PROCS that do not accept a "SAS Variable List" as input and the list must be expanded into individual names.

```
proc sql noprint;
   select _NAME_ into :var separated by ' '
      from &sysmacroname._varlist;
   quit;
   run;
%put NOTE: VAR=&var;
```

The previous example includes PROC SQL to write the values of _NAME_ into macro variable VAR separated by a single space. This technique will suffice for most reasonable sized problems. In the example PROC TRANSPOSE expands the variable list "AGE:" into individual names when the macro is called and the list of variables all exist and are numeric the log looks like the following.

```
790  %main(DATA=sashelp.heart,var=AGE: MRW);

MPRINT(MAIN):   Proc transpose data=sashelp.heart(obs=0 keep=_NUMERIC_) out=MAIN_varlist;
MPRINT(MAIN):   var AGE: MRW;
MPRINT(MAIN):   run;

NOTE: There were 0 observations read from the data set SASHELP.HEART.
NOTE: The data set WORK.MAIN_VARLIST has 4 observations and 2 variables.

NOTE: SYSERR=0
MPRINT(MAIN):   proc sql noprint;
MPRINT(MAIN):   select _NAME_ into :var separated by ' ' from MAIN_varlist;
MPRINT(MAIN):   quit;

MPRINT(MAIN):   run;
NOTE: VAR=AgeCHDdiag AgeAtStart AgeAtDeath MRW
MPRINT(MAIN):   proc summary data=sashelp.heart;
MPRINT(MAIN):   var AgeCHDdiag AgeAtStart AgeAtDeath MRW;
MPRINT(MAIN):   output out=MAIN_summary;
MPRINT(MAIN):   run;

NOTE: There were 5209 observations read from the data set SASHELP.HEART.
NOTE: The data set WORK.MAIN_SUMMARY has 5 observations and 7 variables.

MPRINT(MAIN):   proc print;
MPRINT(MAIN):   run;
NOTE: There were 5 observations read from the data set WORK.MAIN_SUMMARY.
```

Notice that the "SAS Variable List" AGE: is expanded into AgeCHDdiag AgeAtStart AgeAtDeath the entire list being printed on the SAS LOG and used in the VAR statement for PROC SUMMARY. Processing the data from the data set created by PROC TRANSPOSE can be used to produce a more complicated list than blank delimited. For example the values of _NAME_ can be quote and the list can be delimited with a comma using a SELECT statement similar to the one shown here.

```
select quote(strip(_name_)) into :quotedvar separated by ','
         from &sysmacroname._varlist;
```

This produces the following output a comma delimited list of variable names in double quotes.

```
NOTE: QUOTEDVAR="AgeCHDdiag","AgeAtStart","AgeAtDeath","MRW"
```

You can imagine that all kinds of various code bits could be generated in this way.

MANIPULATE A LIST USING ANOTHER LIST.

This example might seem somewhat contrived but I have actually used this exact technique in a macro. A macro was designed to accept a list of covariates (COVARS) to be used as the right hand side of a linear model (main effects ANOVA). The specification was that all COVARS would always be discrete and included in a CLASS statement. The macro was written and put into use, later it was realized COVARS may not always be discrete. The obvious choice would be to add a CLASS parameter to the macro, however since the macro was already in use this posed a compatibility problem. Instead a new parameter DIRECT was added to specify variables that are NOT discrete and should be entered directly into the model. Another similar application was posted to SAS-L in the following SAS-L Thread: Removing STRINGS from A MACRO VARIABLE VALUE . I suggested the method described here but my suggestion was ignored in favor of macro loops and regular expressions. Consider a macro that looks something like the following.

```
%macro main(y=,covars=,direct=);
   proc transpose data=sashelp.heart(obs=0) out=covars;
      var &covars;
      run;
   proc transpose data=sashelp.heart(obs=0) out=direct;
      var &direct;
      run;
   %local class;
   proc sql noprint;
      select _name_ into :covars separated by ' '
            from covars;
      select _name_ into :class separated by ' '
            from covars
         where _name_ ne ALL(select _name_ from direct)
         ;
      quit;
      run;
   %put NOTE: COVARS=&COVARS;
   %put NOTE: CLASS=&CLASS;
%mend main;
```

The first step in the macro is to call PROC TRANSPOSE to produce a variable list for parameters COVARS and DIRECT. This is the same technique we used in previous examples. The error checking done in those examples was omitted from this example to focus attention on this specific task. PROC SQL with the INTO macro variable syntax is used to replace the value of COVARS with the expanded list of variables. A new macro variable CLASS is created by removing the variables found in DIRECT from the list of variables in COVARS.

```
%main
   (
      y      = Diastolic,
      covars = Height Weight Chol_status--Smoking_Status,
      direct = Weight Height
   )

NOTE: COVARS=Height Weight Chol_Status BP_Status Weight_Status
Smoking_Status
NOTE: CLASS=Chol_Status BP_Status Weight_Status Smoking_Status
```

Calling the macro with the options show here will produce the output shown as NOTE. Of course the real program would do something useful with these macro variables, COVARS becoming the right hand side of a MODEL statement and the CLASS becoming the CLASS statement variable list.

Data WORK.COVARS

Obs	_NAME_	_LABEL_
1	Height	
2	Weight	
3	Chol_Status	Cholesterol Status
4	BP_Status	Blood Pressure Status
5	Weight_Status	Weight Status
6	Smoking_Status	Smoking Status

Data WORK.DIRECT

Obs	_NAME_
1	Weight
2	Height

As you can see from the proceeding examples the possibilities for using PROC TRANSPOSE to process variable lists provides a simple and powerful method that should be preferred to manipulation of macro variables. Once the variable names are expanded and put into a SAS data set we can take advantage of other SAS tools designed for manipulation of the data in these data sets.

TRANSPOSE GROUPS TO VARIABLES.

Sometimes it is necessary to transpose groups of observations into variables, where each level of the group becomes a new variable. PROC TRANSPOSE is particularly helpful in this situation because it can create new variables and name them using the values of other variables. The transpose of groups to variables has become more common with the advent of ODS GRAPHIS and the GTL, an example Sample 39132: Median of lipid profile over time illustrates GTL that uses data where levels of treatment are represented by variables. However, the example does not show how to transform the data from the more normal tall structure where treatments are in rows. We will show the transpose of groups to variables is done using SASHELP.CLASS.

Suppose we have data from SASHELP.CLASS and we want to transpose all the numeric variables such that there is a variable for each level of SEX. With three numeric variables (Age, Height, and Weight) and two levels for SEX (M and F) we will create 6 new variables.

```
proc sort data=sashelp.class out=class;
   by sex;
   run;
proc transpose data=class out=class2;
   by sex;
   run;
proc transpose data=class2 out=class3
      DELIMITER=_;
   var col:;
   id sex _name_;
   run;
```

The first step is to sort the data by the grouping variable followed by a transpose of all numeric variables BY SEX. The next step produces a wide data set CLASS2 with one observation for each level of SEX and transposed numeric variable. Remember PROC TRANSPOSE transposes all numeric variables if the VAR statement is not used, as in our example. In step three the wide data is transposed again this time using an ID statement to direct PROC TRANSPOSE to name the new variables using information conveyed through the ID statement. New to SAS 9.2 is the feature to allow more than one variable in the ID statement. The formatted values of the ID variables are concatenated using the delimiter specified in the new parameter DELIMITER =. This new feature in PROC TRANSPOSE saves the extra data step that would be needed to concatenate the values of SEX and _NAME_ into a new ID variable.

Data WORK.CLASS2

Obs	Sex	_NAME_	COL1	COL2	COL3	COL4	COL5	COL6	COL7	COL8	COL9	COL10
1	F	Age	13.0	13.0	14.0	12.0	15.0	11.0	14.0	12.0	15.0	.
2	F	Height	56.5	65.3	62.8	59.8	62.5	51.3	64.3	56.3	66.5	.
3	F	Weight	84.0	98.0	102.5	84.5	112.5	50.5	90.0	77.0	112.0	.
4	M	Age	14.0	14.0	12.0	13.0	12.0	16.0	12.0	15.0	11.0	15.0
5	M	Height	69.0	63.5	57.3	62.5	59.0	72.0	64.8	67.0	57.5	66.5
6	M	Weight	112.5	102.5	83.0	84.0	99.5	150.0	128.0	133.0	85.0	112.0

Data WORK.CLASS3

Obs	_NAME_	F_Age	F_Height	F_Weight	M_Age	M_Height	M_Weight
1	COL1	13	56.5	84.0	14	69.0	112.5
2	COL2	13	65.3	98.0	14	63.5	102.5
3	COL3	14	62.8	102.5	12	57.3	83.0
4	COL4	12	59.8	84.5	13	62.5	84.0
5	COL5	15	62.5	112.5	12	59.0	99.5
6	COL6	11	51.3	50.5	16	72.0	150.0
7	COL7	14	64.3	90.0	12	64.8	128.0
8	COL8	12	56.3	77.0	15	67.0	133.0
9	COL9	15	66.5	112.0	11	57.5	85.0
10	COL10	.	.	.	15	66.5	112.0

The second transpose creates 6 new variables as shown here using the values of SEX and _NAME_, to form the names of the new variables.

DOUBLE TRANSPOSE TO CONVERT NUMERIC VARIABLES TO CHARACTER.

Converting numeric variables to character using PROC TRANSPOSE may seem unusually, I expect most everyone is used to using functions like PUT, PUTN, VVALUE or VVALUEX. Of course those are all well and good but using PROC TRANSPOSE can be much simpler, requiring no data step code. The technique relies on the fact that when PROC TRANSPOSE transposes both character and numeric variables to observations it resolves the type conflict created by this operation by converting numeric variables to character. *The numeric variables are converted using their associated format*, whether it is the SAS default format, a SAS supplied numeric format, or a user written format created with PROC FORMAT. These formats can be specified in a FORMAT statement in PROC TRANSPOSE or permanently associated with the variables in the data set. To see how this technique is applied an a meaningful way see Summary Statistics in Rows where the summary statistics are convert en masse to character efficiently formatting the statistics for display with PROC REPORT.

```
proc format;
   value age 0-12='Pre-Teen' 13-19='Teen';
   run;
data class;
    _OBS_ = _n_;
   set sashelp.class;
   retain _DUMMY_ ' ';
   format age age. weight 7.2 height 7.3;
   label
      DOB='Date of Birth'
      Weight='Weight (kg)'
      Height='Height (in)'
      ;
   run;
```

Consider a data set CLASS created by the statements shown to the left. We will need a unique key variable to identify each observation in a BY statement. This could be a list of variables that uniquely identify the observations but we will create a unique key using _N_. We also need a character variable to transpose along with the numeric variables trigger the conversion. If could be an existing variable but we will use _DUMMY_ to emphasize what is needed to get the desired result. We also add formats to the numeric variables to illustrate that the variables are transposed using their associated formats. Also labels are added to the numeric

variable to show that the labels are also preserved. The first five observations are shown below there are 19 total observations in data CLASS.

Obs	_OBS_	Name	Sex	Age	Height	Weight	_DUMMY_
1	1	Alfred	M	14	69.0	112.5	
2	2	Alice	F	13	56.5	84.0	
3	3	Barbara	F	13	65.3	98.0	
4	4	Carol	F	14	62.8	102.5	
5	5	Henry	M	14	63.5	102.5	

We need to create two variables lists to use in the PROC TRANSPOSE statements VAR and COPY. We need a list of all numeric variables excluding the variable _OBS_ to use in the VAR statement. We also need a list of all character variables excluding _DUMMY_ to use in COPY statement.

```
proc transpose data=class(obs=0 drop=_DUMMY_) out=charvars;
   var _character_;
   run;
proc transpose data=class(obs=0 drop=_OBS_) out=numvars;
   var _numeric_;
   run;
%put NOTE: CHARVARS=&charvars;
%put NOTE: NUMVARS=&numvars;

proc sql noprint;
   select _name_ into :charvars separated by ' '
      from charvars;
   select _name_ into :numvars separated by ' '
      from numvars;
   quit;
   run;
```

These statements should look familiar as they were featured in the examples above. The variable lists are written to macro variables so they can be used in the code that follows.

```
proc transpose data=class
      out=convert2char
      ;
   by _OBS_;
   copy &charvars;
   var &numvars _dummy_;
   run;
```

Given the variable lists created above the first PROC TRANSPOSE is simple. The statement BY _OBS_ transposes each observation separately so the data can be transposed back to its original format. The variables in the CHARVARS macro variable used in the COPY statement and are carried along unchanged. The variable names NUMVARS are transposed along with the character variable _DUMMY_ forcing the entire list of variables into a single variable with character data type.

Obs	_OBS_	Name	Sex	_NAME_	_LABEL_	COL1
1	1	Alfred	M	Age		Teen
2	1			Height	Height (in)	69.000
3	1			Weight	Weight (kg)	112.50
4	1			_DUMMY_		
5	2	Alice	F	Age		Teen
6	2			Height	Height (in)	56.500
7	2			Weight	Weight (kg)	84.00
8	2			_DUMMY_		
9	3	Barbara	F	Age		Teen
10	3			Height	Height (in)	65.300
11	3			Weight	Weight (kg)	98.00
12	3			_DUMMY_		

The variables listed in the VAR statement have been transposed to observations and converted to character using their associated formats and are stored in COL1. The COPY variables have been brought along for the ride. It should be noted that we could COPY any or all numeric variables and if we rename the transposed variables by altering the value of _NAME_ perhaps using
NAME=CATS('C_',_NAME_);
you would end up with the original variables and new variables derived from the numeric variables converted to character with new names.

```
proc transpose
      data=convert2Char
      out=classChar
         (drop=_name_ _dummy_)
      ;
   by _OBS_;
   var col1;
   copy &charvars;
/*   id _name_;*/
/*   idlabel _label_;*/
   run;
```

To get the data back to its original format we can use the PROC TRANSPOSE show here. Again we use BY unique record identifier to identify which groups of observations to collapse into one observation. This time the variable COL1 is transposed and the same list of tagalong variables are listed in the COPY statement. Recall the special variables _NAME_ and _LABEL_ imply the inclusion of the ID and IDLABEL statements, show here commented out of the program.

Variables in Creation Order				
#	Variable	Type	Len	Label
1	_OBS_	Num	8	
2	Name	Char	8	
3	Sex	Char	1	
4	Age	Char	8	
5	Height	Char	8	Height (in)
6	Weight	Char	8	Weight (kg)

This PROC CONTENTS shows that AGE, HEIGHT, and WEIGHT have been converted to character. Plus the variable LABEL remains attached. Looking at the next table of 4 observations displayed with PROC PRINT we can see how Age was converted to a text string using the AGE format and HEIGHT and WEIGHT while more subtle have been converted similarly using their associated formats.

Obs	_OBS_	Name	Sex	Age	Height	Weight
1	1	Alfred	M	Teen	69.000	112.50
2	2	Alice	F	Teen	56.500	84.00
3	3	Barbara	F	Teen	65.300	98.00
4	4	Carol	F	Teen	62.800	102.50

DOUBLE TRANSPOSE TO CONVERT CHARACTER VARIABLES TO NUMERIC.

Suppose we need to convert a large number of character variables to numeric. As you know we can't just change a variable's data type we must create a new variable. Also if we want the new variables to have the same name and label we have to RENAME and DROP and reassign the LABEL. This can be a bit tedious but we could write a macro to do the RENAME and DROP and the rest. Or PROC TRANSPOSE can be employed making it all fairly easy. Consider the following example that address the question posted to the SAS Discussion Forum "convert char to num for all _CHARACTER_"

```
data test;
   id + 1;
   input (x1-x10)(:f8. :$f8.);
   attrib _numeric_
      label='Numeric Variable';
   attrib _character_
      label='Character Variable';
   cards;
1  2  3  4  5  6  7  8  9 10
3 49 98 17 87 18 18 18 77 77
;;;;
   run;
```

The data consists of a unique ID variable and some numeric and character variables. There could be any number of variables too many to deal with individually. In this example the range of variables to be processed is X1-X10.

```
proc transpose data=test(obs=0) out=all;
   var _all_;
   run;
proc transpose data=test(obs=0) out=nums;
   var x1-numeric-x10;
   run;
proc transpose data=test(obs=0) out=chars;
   var x1-character-x10;
   run;
proc sql noprint;
   select _name_ into :all separated
      by ' ' from all;
   select _name_ into :nums separated
      by ' ' from nums;
   select _name_ into :chars separated
      by ' ' from chars;
   quit;
   run;
%put NOTE: ALL=&all;
%put NOTE: NUMS=&nums;
%put NOTE: CHARS=&chars;
```

We will create three variable lists using the technique of transposing zero observations described above. ALL will contain a list of all variables in the order of the original data. NUMS are the names of the numeric variables in the range of variables of interest. CHARS are the names of the character variables in the range of interest. Each variable list is written to macro variables of the same name. The macro variables are used in the next steps.

```
NOTE: ALL=id x1 x2 x3 x4 x5 x6 x7 x8 x9 x10
NOTE: NUMS=x1 x3 x5 x7 x9
NOTE: CHARS=x2 x4 x6 x8 x10
```

```
proc transpose data=test out=flipchar;
   by id;
   var &chars;
   copy &nums;
   run;
data flipchar;
   set flipchar;
   num = input(col1,f16.);
   run;
```

Armed with the variable lists created above we can flip the character variables to long format by the unique ID and drag along the COPY varriables. With a simple data step we add a new variable to FLIPCHAR with an assignment statement using an INPUT function we convert COL1 to numeric. Effectively converting all character variables to numeric, as soon as we flop the data back to wide that is.

Obs	id	x1	x3	x5	x7	x9	_NAME_	_LABEL_	COL1	num
1	1	1	3	5	7	9	x2	Character Variable	2	2
2	1	.	.	.	.	.	x4	Character Variable	4	4
3	1	.	.	.	.	.	x6	Character Variable	6	6
4	1	.	.	.	.	.	x8	Character Variable	8	8
5	1	.	.	.	.	.	x10	Character Variable	10	10
6	2	3	98	87	18	77	x2	Character Variable	49	49
7	2	.	.	.	.	.	x4	Character Variable	17	17
8	2	.	.	.	.	.	x6	Character Variable	18	18
9	2	.	.	.	.	.	x8	Character Variable	18	18
10	2	.	.	.	.	.	x10	Character Variable	77	77

Notice the variables in macro variable NUMS have been copied to the new data set via the COPY statement. Also note the new variable NUM; this will be the variable that is transpose when the data is transposed back to wide format.

```
proc transpose data=flipchar
      out=flop2allnums(drop=_name_);
   by id;
   var num;
   *id _name_;
   *idlabel _label_;
   copy &nums;
   run;
```

Data FLIPCHAR is now transposed back to wide format. The transpose variable is NUM and since the special variables _NAME_ and _LABEL_ exist in the input they are used in implied ID and IDLABEL statements, providing variable names and labels for the new numeric variables created by the transpose to wide. The COPY statement copies the original COPY variables to this new data set.

Obs	id	x1	x3	x5	x7	x9	x2	x4	x6	x8	x10
1	1	1	3	5	7	9	2	4	6	8	10
2	2	3	98	87	18	77	49	17	18	18	77

The new data FLIP2ALLNUMS now contains new numeric variables converted from character and the original variables, but the variable order is now different.

```
data allChars;
   retain &all;
   set flop2Allnums;
   run;
```

We can fix the order using the macro variable ALL and a RETAIN statement in a data step.

Variables in Creation Order			
Variable	Type	Len	Label
id	Num	8	Numeric Variable
x1	Num	8	Numeric Variable
x2	Num	8	Character Variable
x3	Num	8	Numeric Variable
x4	Num	8	Character Variable
x5	Num	8	Numeric Variable
x6	Num	8	Character Variable
x7	Num	8	Numeric Variable
x8	Num	8	Character Variable
x9	Num	8	Numeric Variable
x10	Num	8	Character Variable

At last we have produced the data set where specific character variables are converted to numeric variables with all variables in the original order and with the variable label from the original variable.

CONCLUSION

The examples presented here demonstrate a few useful techniques that are available using proc PROC TRANSPOSE. PROC TRANSPOSE is invaluable for manipulating variable lists as it provides the most efficient method I have found to expand a list and put the output into an infinitely useful object.

REFERENCES

- SAS Variable Lists http://support.sas.com/documentation/cdl/en/lrcon/61722/HTML/default/a000695105.htm
- Summary Statistics in Rows http://www.lexjansen.com/mwsug/2011/pharma/MWSUG-2011-PH02.pdf
- The TRANSPOSE Procedure http://support.sas.com/documentation/cdl/en/proc/61895/HTML/default/viewer.htm - a000063661.htm

CONTACT INFORMATION

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

John Henry King
Ouachita Clinical Data Services, Inc.
1769 Hwy 240 West
Hopper, AR 71935
PH: 501-351-0432
iebupdte@gmail.com