An Animated Guide: Proc Transpose
Russell Lavery, Independent Consultant

ABSTRACT
If one can think about a SAS® data set as being made up of columns and rows one can say Proc Transpose flips the columns of data into rows of data -and the reverse. The ability to “flip” data is necessary because some SAS Procs want data to “be in rows” and some Procs want the data to “be in columns”. Complex manipulations of the shape of the data (tall & thin data sets into wide and short data sets and reverse) can be quickly performed with Proc Transpose. The trick in using Proc Transpose is to recognize that the statements in your code do not completely determine the output you get. Output is determined by your individual coded statements, the interaction between coded statements, the shape of the input data set and also by actions that Proc Transpose takes by default.

INTRODUCTION
This paper will contain many slides and it should be noted that each slide is a complete worked example and a self-contained illustration of a particular point.

Many programmers, when they have to “reshape” data sets, use a data step and justify their choice by saying that they want control of the process. Proc Transpose, properly understood, also gives control. Anyone doing reporting on multiple levels (e.g. reporting on region and state and county and city and zip) will find the combination of Proc Summary output files “pushed through” a Proc Transpose to be a great time saver.

Three major issues must be overcome in understanding Proc Transpose. The first is that coded statements/options interact to produce the output. This means that understanding statement combinations- not statements- is required. The second issue is that, for the same Proc Transpose code, input data files with different shapes can produce output data files with different shapes. The third issue is that Proc Transpose has several actions that it takes on its own - in the absence coded statements. This means that if the programmer does not code statements to tell Proc Transpose what to do, Proc Transpose will take action on its own. These automatic actions can interact with coded instructions.

The basic syntax (no Let or Copy) of Proc Transpose is shown in Figure 1 and the basic process is shown in Figure 2. This paper will endeavor to cover, in detail, the statements in Figure 1. Let and Copy statements (that are not shown in Figure 1) will be briefly covered in Section 5.
GOALS/ISSUES
A useful structure for the task of mastering Proc Transpose is to break the material into five sections/issues/goals. This paper uses these sections to organize the material. The sections are:

Section 1 - Understanding which variables will be transposed
Section 1 - Understanding the storing the old variable names in columns in the new dataset – automatically and manually
Section 3 - Understanding the creation of names and labels for the variables being created – maintaining a link to reality
Section 4 - Understanding how to control the shape of the output data set (rows and columns)
Section 5 - Understanding a few quirks: Let, Copy, By combined with ID

Note, in Figure 2, the numbers in the source data set (class) had “links to reality” by means of the stu_id column and the names of the variables. There is one 83 score in the data and it is the score of Stu_Id 3 on test 2. These “links to reality” were destroyed by the transposition in Figure 2. The 83 is in the output data set but we no longer have, in the data set itself, any way of linking the 83 to reality (linking to a student and test). This data set is small and the observations can be “tracked” by visual examination. If the data set had thousands of observations, a visual checking would be difficult. We need to have Proc Transpose maintain links to reality as it does the transposing.

SECTION 1
WHICH VARIABLES WILL BE TRANSPOSED
The var statement specifies the variables to be transposed and is illustrated in Figure 2. As you can see, the rows in the source data set have been “flipped” into columns in the output data set. Row 1 in Class is turned into column 1 in For_pres. Column 1 in Class is now row 1 in For_pres. This “flipping” is the basic operation of a transpose. However, with just this basic operation, it is difficult to look at the output data set and link a score to a student or test. There are no links to reality in the output data set.

SECTION 2
STORING THE OLD VARIABLE NAMES IN COLUMNS IN THE NEW DATASET– AUTOMATICALLY AND MANUALLY
A key technique to maintaining links to reality through the transpose is to store the old variable names in the new data set. This is so useful that Proc Transpose will do it as a default. This process is shown in Figures 3 and 4.

Figure 3 shows the automatic creation of two variables (_name_ and _label_) to hold information about the source data set. They hold links to reality. (Note that the code in Figure 3 only does half the job—there is no student information in the outfile). By default, Proc Transpose will create 1 or 2 new columns (_name_ always
and _label_ if the variables have labels ), in the output data set. By default the names of these variables are _name_ and _label_. Rows in these new variables/columns contain the names and labels of the columns in the original data set. Remember a column in the old data set turns into a row in the new data set, so it makes sense to store the name of the old column in a row of the new data set.

Figure 4 shows the second half of a "double transpose". Figure 3 shows a transpose of the data set class. Figure 4 shows the transpose of the output data set created in Figure 3… a double transpose… or a transpose of a transpose. Note that Proc Transpose automatically accesses the values in _name_ and _label_ and uses them to create names and labels for variables in the output data set. This is an important rule of transpose. Without you coding any instructions, Proc Transpose will use variables called _name_ and _label_ if they exist in the input data set.

The Proc Print in Figure 4 uses the variable names as column headers. If that Proc Print in Figure 4 statement had the label option Figure 4 would show that values in the label statement had been assigned as labels for the columns in the output data set.

Please pause and consider three issues. Firstly; most people would hope that a transpose of a transpose would get them back to the original data file; back to where they started from. In Figure 4 we see that it did not. The student information has been lost. It is possible for a double transpose to keep track of student information. It is good programming practice and is easy to do. It simply requires some commands to be seen later in the paper. Secondly; as all transposes like to do, the second transpose also created _name_ as it created data set Class1b. Thirdly, as we see in Figure 4, the value of Test 1 in a row in _name_, in file ForPresb, will be applied to a column in the output data set class1b. You must be sure, in your more complicated transposes, that the code will attempt to “put” that value of _name_ over only one column, or the transpose will fail.

### SECTION 3

**CREATING NAMES AND LABELS FOR THE VARIABLES BEING CREATED – MAINTAINING A LINK TO REALITY**

Figures 4 and 5 showed part of the technique for maintaining links to reality through a transpose. These figures show how to store the names and labels of the old variables in variables (_name_ _label_) in the new data set. This is half of the process of maintaining links to reality.

An examination of Figures 2 and 3 shows several new variables (with names like col—col3) that hold the transposed data. The names col1—col3 are not very informative. If one looks at Figure 3 it can be seen that all the data for Student 001 ended up in col1. It would be desirable to name that column something like Student_1.
If that were the possible, a programmer would be able to maintain both row and column “links to reality” through a transpose. In fact, this is possible and the technique will be shown below.

As transpose creates variables to hold the transposed data, the name of the new variables is the result of concatenating two pieces (call them left and right) of information into one column name. The rule for name creation is that the value in the prefix= option is the left hand part and the value in the ID statement is the right hand part of the name. Prefix= specifies a string and that string does not change. This makes prefix= ideal for creating columns with a constant left hand side (like student_1 student_2). The right hand side of the name is specified by the ID statement and points to a variable in the data set. This makes ID ideal for making column names with a varying right hand side (like student_1, student_2, Territory_NY, Territory_NJ).

The eagerness of Proc Transpose to be helpful, and take action without instructions, makes this coding a bit confusing. If you specify prefix= and ID, Proc Transpose follows instructions and assembles the column names. However, you can specify, or not specify, either of the options. When either right, or left, is omitted Proc Transpose will take the initiative to “help” create logical and useful names.

### Transpose Issue

**Rules for Manually Creating Var. Names**

VarName = part1 || part2

- **part1=prefix**
- **part2=id**

<table>
<thead>
<tr>
<th>No Part1</th>
<th>No Prefix Option</th>
<th>Part1=prefix</th>
<th>Part2=id</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Part1</td>
<td>No Prefix Option</td>
<td>Part1=prefix</td>
<td>Part2=id</td>
</tr>
<tr>
<td>Col1 to Coln</td>
<td>State 1</td>
<td>Prefex1 to Prefix2</td>
<td>State 2</td>
</tr>
<tr>
<td>ID</td>
<td>State 3</td>
<td>Prefix</td>
<td></td>
</tr>
</tbody>
</table>

**No default for missing prefix**

**Default is n**

**Figure 7**

**Figure 8**

A programmer can specify prefix=, or not. A programmer can specify ID, or not. Combining the ways of specifying the two instructions gives the four states of nature shown in Figure 7. Figure 7 shows the states of nature that exist and the rules that Proc transpose will apply for each of the four states.

Figure 7 says that if a programmer does not specify either Prefix or ID (state of nature 1), the column names will be assembled as Col1 to Coln (Column||N). Transpose automatically makes “col” the value of the left hand part of the name and an increasing series (1,2,3) the value of the right hand part of the name. This is shown in Figure 3.

If only the Prefix= option is specified (state of nature 2), the name will be Prefix1 to Prefixn. Again, Transpose automatically makes an increasing series (1,2,3) the value of the right hand part of the name. This is what has happened in Figures 5 and 6, though, with only one column, the pattern is not obvious.

If the prefix= option is not specified and the ID statement is (state of nature 3), the column names will be taken from the ID variable in the old table. The values of ID are the new names. Proc Transpose does not assume any value for the left part of the new name and there will be no 1,2,3 etc. as the right hand part of the column name. This is shown in Figure 8 and can be quite useful if the ID variable is character.
State of nature 4 offers great flexibility in creating names. If the ID variable is numeric, as is shown in Figure 10, use of a prefix allows the programmer to create names for the new columns that are both valid SAS names (not starting with a number) and preserve the link to reality. Having Proc Transpose maintain the link to reality, as opposed to “fixing names” in a post-transpose data step, is efficient and reduces mistakes. Figures 9 and 10 illustrate the use of Prefix= and ID in combination.

Sometimes the values to be used to create labels and names are not in a variable named _LABEL_ (the standard source for label information). Figure 11 shows how to tell Proc Transpose to get the label information from a variable s/he chooses and not from the automatic variable _Label_. The IDLABEL statement allows a programmer to tell Proc Transpose to get variable information (source for label information). Figure 11 shows how to tell Proc Transpose to get the label information from a custom named variable. The IDLABEL statement allows a programmer to tell Proc Transpose to get variable label information from a variable s/he chooses and not from the automatic variable _Label_.

**SECTION 4**
CONTROLLING THE SHAPE OF THE OUTPUT DATA SET (ROWS AND COLUMNS)
As a Q.C. measure, and a guide to proper coding, it is useful to be able to predict how Proc Transpose statements will change the “shape” of a data set. This ability to predict the output shape helps a programmer decide what statements to code. Actually, the programmer usually knows what he wants and what the data looks like and must imagine how to code the Proc Transpose statements to get the desired output. This section examines how the data set changes shape in response to the BY and VAR statements.

Some rules for predicting the shape of the transpose are:
- For a simple transpose think that:
  - St_ID  OLD_VAR  OLD_LBL  Score
  - St_ID  _NAME_   E1    E2          St_ID      VARIABLE    Exam1 Exam2

Obs  _NAME_  _LABEL_  Stu_27       Stu_28      Stu_29
28        40        55         .    Y
27        92        83      88    Z
29        92        83      88    Z
27        77        78      99    X
26        92        83      88    Z
25        92        83      88    Z
24        89        78      99    X
23        92        83      88    Z
22        89        78      99    X
21        67        58      99    X
20        77        78      99    X
19        92        83      88    Z
18        92        83      88    Z
17        89        78      99    X
16        89        78      99    X
15        89        78      99    X
14        89        78      99    X
13        89        78      99    X
12        89        78      99    X
11        89        78      99    X
10        89        78      99    X
9         89        78      99    X
8         89        78      99    X
7         89        78      99    X
6         89        78      99    X
5         89        78      99    X
4         89        78      99    X
3         89        78      99    X
2         89        78      99    X
1         89        78      99    X

Obs  _NAME_  _LABEL_  Stdnt_X    Stdnt_Y    Stdnt_Z
28        40        55         .    Y
27        92        83      88    Z
29        92        83      88    Z
27        77        78      99    X
26        92        83      88    Z
25        92        83      88    Z
24        89        78      99    X
23        92        83      88    Z
22        89        78      99    X
21        67        58      99    X
20        77        78      99    X
19        92        83      88    Z
18        92        83      88    Z
17        89        78      99    X
16        89        78      99    X
15        89        78      99    X
14        89        78      99    X
13        89        78      99    X
12        89        78      99    X
11        89        78      99    X
10        89        78      99    X
9         89        78      99    X
8         89        78      99    X
7         89        78      99    X
6         89        78      99    X
5         89        78      99    X
4         89        78      99    X
3         89        78      99    X
2         89        78      99    X
1         89        78      99    X

Obs  _NAME_  _LABEL_  Stu_27       Stu_28      Stu_29
28        40        55         .    Y
27        92        83      88    Z
29        92        83      88    Z
27        77        78      99    X
26        92        83      88    Z
25        92        83      88    Z
24        89        78      99    X
23        92        83      88    Z
22        89        78      99    X
21        67        58      99    X
20        77        78      99    X
19        92        83      88    Z
18        92        83      88    Z
17        89        78      99    X
16        89        78      99    X
15        89        78      99    X
14        89        78      99    X
13        89        78      99    X
12        89        78      99    X
11        89        78      99    X
10        89        78      99    X
9         89        78      99    X
8         89        78      99    X
7         89        78      99    X
6         89        78      99    X
5         89        78      99    X
4         89        78      99    X
3         89        78      99    X
2         89        78      99    X
1         89        78      99    X

Obs  _NAME_  _LABEL_  Stu_27       Stu_28      Stu_29
28        40        55         .    Y
27        92        83      88    Z
29        92        83      88    Z
27        77        78      99    X
26        92        83      88    Z
25        92        83      88    Z
24        89        78      99    X
23        92        83      88    Z
22        89        78      99    X
21        67        58      99    X
20        77        78      99    X
19        92        83      88    Z
18        92        83      88    Z
17        89        78      99    X
16        89        78      99    X
15        89        78      99    X
14        89        78      99    X
13        89        78      99    X
12        89        78      99    X
11        89        78      99    X
10        89        78      99    X
9         89        78      99    X
8         89        78      99    X
7         89        78      99    X
6         89        78      99    X
5         89        78      99    X
4         89        78      99    X
3         89        78      99    X
2         89        78      99    X
1         89        78      99    X

More Generally:
You get a transpose for each “combination of the levels” in variables in the BY statement.

The number of rows in the output data set = # of vars transposed * levels of BY.
If no BY variables, SAS assumes one level of BY.

The number of columns in the output data set = # of New vars + rows in BY + By Vars + auto. Vars.
If no BY statement, SAS assumes one level of BY.
Automatic vars. are _name_ and _label_.

The shape of the transpose is controlled by the interaction of the VAR and BY statements. A programmer can specify either one, or more than one variable in the VAR statement. A programmer can specify either none, one, or more than one variable in the BY statement. Combining the ways of specifying the two instructions gives the six states of nature shown in Figure 12. Figure 12 shows the states of nature we must understand and the rules for each of the four states will be developed below. Once again we must examine several states of nature.

State of nature one is shown in Figure 16 (and others).

A mental crutch is to imagine laying rectangles over the data set. In the Figures illustrating these six states of nature, the boxes have been drawn. Draw a rectangle around the variables in the var statement (column(s)). Draw a rectangle around one combination of values of the variable(s) in the BY statement (row(s)). The box where the two rectangles intersect will show the transpose that will take place. You get a transpose for every combination of levels of variables in the BY statement. If there is no BY statement, make the rectangle enclose all the rows in the data set, as is shown in Figure 16.

Figure 16 gives odd looking results until one realizes that the code is transposing a 3 by 1 matrix and that the output of a 1 by 3 matrix. Note that there is a transpose within each level of the by variable. Think that there is only one level of the BY. A column has become a row.

State of nature two is illustrated in Figure 17. In Figure 17 we see a request to transpose a 3 by 3 matrix (t1 t2 t4 can be considered a disjointed 3 by 3 matrix). This will produce a 3 by 3 matrix.
State of nature three is illustrated in Figure 18. Think of boxes layered on the data set. One box is “defined by a layer of the BY variable and the other box by the vars in the var statement. Where the boxes overlap is the area that will be transposed. In Figure 18 we see a request to transpose a 1 by 1 matrix. This will produce a 1 by 1 matrix for each level of the by variable.

Figure 19 illustrates state of nature four and the use of Proc Transpose on data from Proc Summary. Figure 19 illustrates the effect of multiple variables in the var statement. The code asks for the transpose of a 5 by 2 matrix for each level of the by variable. The output is several 2 by 5 arrays.

State of nature five is illustrated in Figure 20. Figure 20 illustrates the effect of multiple variables in the BY statement and one var in the var statement. A transpose is done for each “combination of levels” of the variables in the BY (a transpose for F-11, F-12…..M-15, M16). Each combination of levels gets a transpose and you can see that the overlapping rectangles define a 1 by 1. The code asks for the transpose of a 1 by 1 matrix for each combination of levels of variables in the BY. Two examples of using rectangles to illustrate the transpose have been drawn on Figure 20.

State of nature six is illustrated in Figure 21. Figure 21 illustrates the effect of multiple variables in the BY statement and multiple vars in the var statement. A transpose is done for each “combination of levels of the variables in the BY. Each combination level gets a transpose and you can see that the combination level defines a 2 by 1. The code asks for the transpose of a 2 by 1 matrix for each “combination level” of the variables in the BY. Two examples of using rectangles to illustrate the transpose have been drawn on Figure 20.
SECTION 5

A FEW QUIRKS: MIXED CHAR AND NUMERIC VARS, LET, COPY, BY COMBINED WITH ID

Figure 22 illustrates an unfortunate limitation in the logic of transposing. Using the “rectangle crutch” (not shown) a level of the BY and the two var var would define a 4 by 2. The transpose is a 2 by 4 for each level of st_id. The code asks for a transpose of col1 (a numeric var) and sex (a character var). These variables, after transpose, are in the same column and are BOTH transformed to character with odd alignments. Often the char/numeric issue forces the programmer to do two Proc Transpose s (one with the char variables and one with the numeric) followed by a data set merge.
Figure 23 illustrates the use of the copy statement. This statement copies one, or more, variables from an input data set to an output data set without transposing them.

Figure 24 illustrates the let statement. PROC TRANSPOSE transposes the observation that contains the **LAST OCCURRENCE** of a particular ID value within the DATA SET or the BY GROUP.

**SUMMARY**

Proc Transpose has simple rules and is very useful to SAS programmers, especially when used on output from Proc Summary. Programmers should maintain links to reality by use of proper SAS code.

**CONTACT INFORMATION**

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

Russel Lavery,
Independent Consultant
Russ.Lavery@verizon.net

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.