# Paper 171-2008

# Introduction to PROC TABULATE

Thomas J. Winn, Jr. U.S. Department of Veterans Affairs, Austin, Texas

#### **ABSTRACT**

PROC TABULATE is used to build tabular reports containing descriptive statistical information, including hierarchical relationships among variables. The code that is used to invoke PROC TABULATE is complicated, and much of it looks quite different from other SAS procedures. Nevertheless, it is well worth the necessary investment of time and effort for learning the intricacies and subtleties of its syntax. This paper provides a simplified, step-by-step approach for coding PROC TABULATE.

#### INTRODUCTION

PROC TABULATE is used to build tabular reports containing descriptive statistical information, including hierarchical relationships among variables. PROC TABULATE is the SAS® System's implementation of TPL (Table Producing Language), which was developed at the U.S. Bureau of Labor Statistics during the 1970s, for generating tabular reports of descriptive statistics involving employment data.

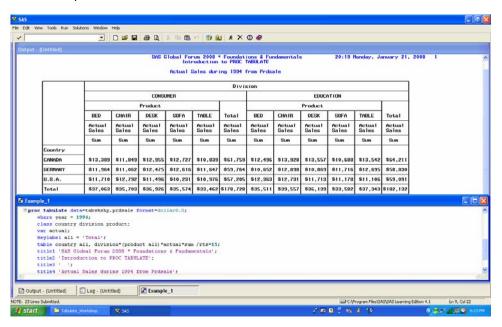
PROC TABULATE is more powerful for producing tabulations than PROC FREQ, and it is a more flexible statistical report writer than PROC MEANS. Although PROC TABULATE and PROC REPORT are both capable of generating similar tabular reports in many situations, each of these procedures has strengths and weaknesses. PROC TABULATE seems to be better for displaying hierarchical relationships. The syntax used to invoke PROC TABULATE and PROC REPORT differ from one another, and both are complicated. However, it is well worth the necessary investment of time and effort for learning the intricacies and subtleties of coding <a href="https://example.com/brocedures">both</a> procedures. This paper will cover the fundamentals of coding PROC TABULATE.

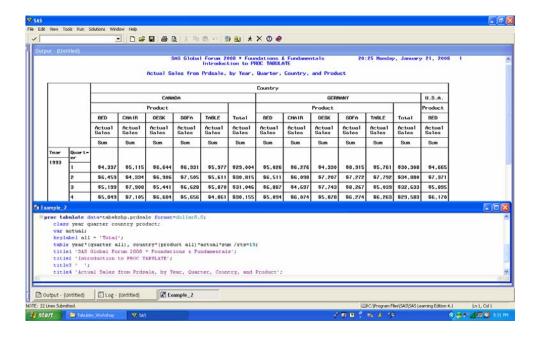
The examples in this paper make use of the SAS® data files, *prdsale* (from the SAS 9.1 <u>sashelp</u> data library), and *empldata* (an inner join of the *empinfo*, *jobcodes*, and *salary* SAS® data sets, from the SAS 9.1 <u>sample</u> data library).

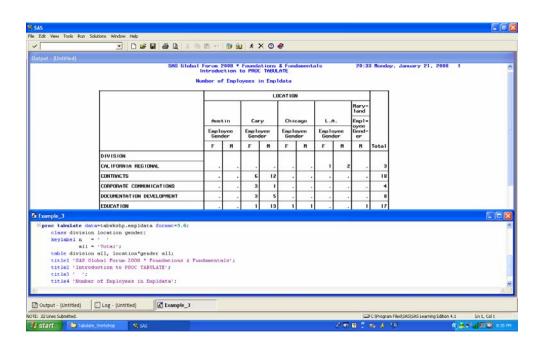
# **GETTING STARTED WITH PROC TABULATE**

The first thing the would-be PROC TABULATE programmer needs to do, *before* beginning to write any SAS code, is to decide what the final report should look like. What is the intended design for the pages, rows, and columns of the report?

Here are a few examples of uses of PROC TABULATE:







Unfortunately, a lot of the code that is used to invoke PROC TABULATE looks quite different from the code that is used for other SAS procedures. Here is the basic syntax for coding PROC TABULATE:

```
PROC TABULATE <option-list>;
     BY <NOTSORTED> <DESCENDING> variable-1
           < ... <DESCENDING> variable-n>;
     CLASS class-variable-list;
    CLASSLEV class-variable-list / STYLE=<style-element-name>
                         <[style-attribute-specification(s)]>;
    VAR analysis-variable-list;
    TABLE <<pre><<pre>row-expression,>
                  column-expression </ table-option-list>;
     FORMAT variable-list-1 format-1
                     < ... variable-list-n format-n>;
    FREQ variable;
    KEYLABEL keyword-1='description-1'
                         < ... keyword-n="description-n'>;
    KEYWORD keyword(s) / STYLE=<style-element-name>
                         <[style-attribute-specification(s)]>;
     LABEL variable-1='label-1' <...variable-n='label-n'>;
    WEIGHT variable;
    TITLE 'text';
```

Here are some options that are frequently used in PROC TABULATE statements:

```
DATA=SAS-dataset-name
FORMAT=formatname
MISSING
NOSEPS
```

The DATA= option specifies the SAS dataset to be used.

The FORMAT option specifies a default format for each table cell. The default format is overridden by any format specified in a subsequent TABLE statement.

The *MISSING* option requests that missing values be regarded as valid levels for classification variables. Unless the MISSING option is specified, observations with missing values for class variables will not be included in the analysis.

The NOSEPS option removes the interior horizontal lines from the printed report.

### STEPS FOR CODING PROC TABULATE

The simplest way to approach coding for PROC TABULATE may be expressed in terms of the following five steps:

- (1) Specification of classification variables,
- (2) Specification of analysis variables,
- (3) Definition of dimensions of the table,
- (4) Identification of desired statistics, and
- (5) Labeling and formatting the table.

### WHAT ARE CLASSIFICATION AND ANALYSIS VARIABLES?

<u>Classification variables</u> are used to identify categorical groups on which calculations are performed. They are the variables that make up the rows and columns of the table. They may be either character or numeric. If numeric, generally only a small number of distinct values are permitted.

<u>Analysis variables</u> are numeric variables that are used to compute statistics that are reported in the body of the table.

### STEP 1 - SPECIFICATION OF CLASSIFICATION VARIABLES

The CLASS statement is used to specify any categorical variables that will be used for grouping purposes in the analysis. The CLASS statement is required.

```
PROC TABULATE DATA=tabwkshp.prdsale;
CLASS country division product; . . .
```

### STEP 2 - SPECIFICATION OF ANALYSIS VARIABLES

The *VAR* statement is used to list any variables that will be used for computing the statistics that are to appear in the body of the table. Frequency counts can be computed without an analysis variable. However, under most conditions, the *VAR* statement is required.

```
PROC TABULATE DATA=tabwkshop.prdsale;
CLASS country division product;
VAR actual; . . .
```

### STEP 3 - DEFINITION OF THE TABLE'S DIMENSIONS

The *TABLE statement* is used to define both the arrangement of the rows and columns of the table, as well as the requests for any summary statistics. It is the tricky aspect of using PROC TABULATE. So let's consider this complex statement one piece at a time.

In a TABLE statement, the comma is a very important symbol, because it separates the dimensions of the table.

- If <u>two</u> commas were specified, then the table would have <u>three</u> dimensions, and the order would be <u>pages</u>, <u>rows</u>, and <u>columns</u>.
- If only <u>one</u> comma was specified, then the table would have <u>two</u> dimensions, and the order would be <u>rows</u>, <u>columns</u>.
- No comma would be interpreted to mean that the table's *only* dimension would be the *column* dimension. The table would only have one row.

So, TABLE statements look like the following:

```
TABLE page-expression, row-expression, column-expression . . .;
TABLE row-expression, column-expression . . .;
or
TABLE column-expression . . .;
```

In this context, an <u>expression</u> can consist of variables, statistics, operators, format specifications, and label assignments.

Because our immediate concern is only to define the table's *dimensions*, we are pleased to discover that only three operators are needed to specify the page, row, and column headings that identify the structure of a table.

- An asterisk (\*) can be used to <u>cross</u> the classification variables; that is, to arrange them in a nested manner, according to the order listed (top, middle, and lower).
- A blank space is used to <u>concatenate</u> two classification variables (which will appear in the table: top-to-bottom for row headings, left-to-right for column headings).
- Parentheses () are used to <u>group</u> the elements of an expression, and to associate an adjacent operator with each concatenated element inside the parentheses.

Here are a few simple examples:

```
TABLE var1, var2;
```

The preceding would generate a two-dimensional table in which the row dimension would be the values of *var1*, and the column dimension would be the values of *var2*.

```
TABLE var1, var2 var3;
```

This would result in a two-dimensional table in which the row dimension would be the values of *var1*, and the column dimension would be comprised of values resulting from the side-by-side concatenation of *var2* and *var3*.

```
TABLE var1, var2*var3;
```

This statement would generate a two-dimensional table in which the row dimension would be the values of *var1*, and the column dimension would be a hierarchical arrangement of the values of *var2* and *var3*, with the values of *var2* comprising the top columns, and for each of these values as columns, the values of *var3* as the lower columns.

Here are three additional examples:

```
TABLE var1, var2 * (var3 var4) var5;
```

The preceding would generate a two-dimensional table in which the row dimension would be the values of *var1*, and the column dimension would be two side-by-side components, the first of which would be a hierarchical nesting of the values of *var2* over a side-by-side arrangement of the values of *var3* and *var4*, and the second component would be the values of *var5*.

```
TABLE var1, (var2 var3) * var4;
```

This would result in a two-dimensional table in which the row dimension would be the values of *var1*, and the column dimension would be two side-by-side hierarchical components, the first of which would be the values of *var2* over the values of *var4*, and the second would be the values of *var3* over the values of *var4*.

```
TABLE var1 var2, var3 * var4;
```

This code would generate a two-dimensional table in which the row dimension would be the concatenated values of *var1* and *var2*, and the column dimension would be a hierarchical nesting of the values of *var3* over the values of *var4*.

#### STEP 4 - IDENTIFICATION OF DESIRED STATISTICS

We have been considering incomplete *TABLE* statements (because up to this point we have only focused on the *dimensions* of the tables).

Besides specifying the dimensions, the *TABLE* statement also identifies which <u>summary statistics</u> should be produced, and pertaining to which analysis variables. Each statistic is identified by a *keyword*.

N = the number of observations, the frequency count

MIN = the smallest value

MAX = the largest value

MEAN = the arithmetic mean, or the average value

*STD* = the standard deviation

VAR = the variance

MEDIAN = the middle (50th percentile) value

SKEWNESS = a measure of the asymmetry of the distribution of values

SUM = the sum of the values

PCTN = the percentage that one frequency is of another frequency

PCTSUM = the percentage that one sum is of another sum

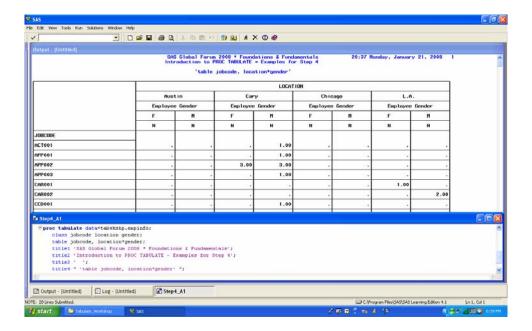
. . . (and other descriptive statistics).

Whenever you cross a variable with a keyword for a statistic, you are identifying the statistic to be applied to that variable (which tells PROC TABULATE what type of calculation to perform). You can cross classification variables only with the *N* or *PCTN* statistics. By default, if the *TABLE* statement does not include an analysis variable or a statistic, then PROC TABULATE automatically crosses the *N* statistic with the indicated class variables. Analysis variables can be crossed with any statistic. By default, if the *TABLE* statement includes an analysis variable but without crossing it with any statistic, PROC TABULATE automatically crosses it with *SUM*.

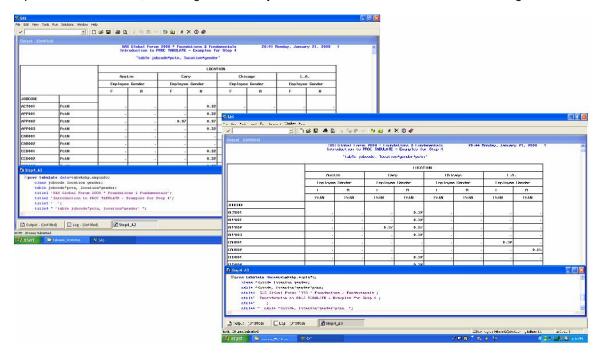
Here are some examples:

```
PROC TABULATE DATA=tabwkshp.empldata;
CLASS jobcode location gender;
TABLE jobcode, location*gender;
TABLE jobcode*PCTN, location*gender;
TABLE jobcode, location*gender*PCTN;
TABLE jobcode PCTN, location* gender;
```

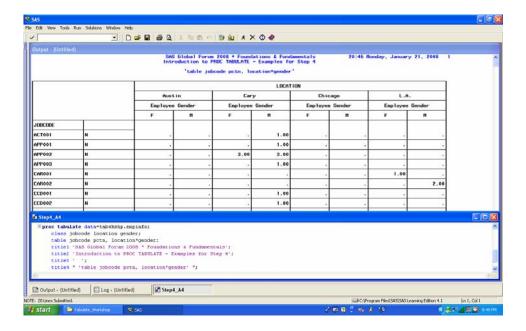
The first *TABLE* statement would generate a hierarchical breakdown of frequency counts in the data set, according to values of *jobcode* (the rows) and the nested values of *location* and *gender* (the columns).



The second and third *TABLE* statements would generate a hierarchical breakdown of percentages represented in each cell, according to values of *jobcode* and the nested values of *location* and *gender*.



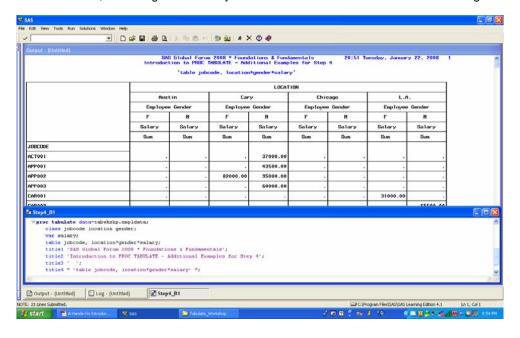
The fourth *TABLE* statement would generate a hierarchical breakdown of frequency counts represented in each cell, according to values of *jobcode* and the nested values of *location* and *gender*. It would include an additional row that would represent the percentage of the total population of the data set included in each column.



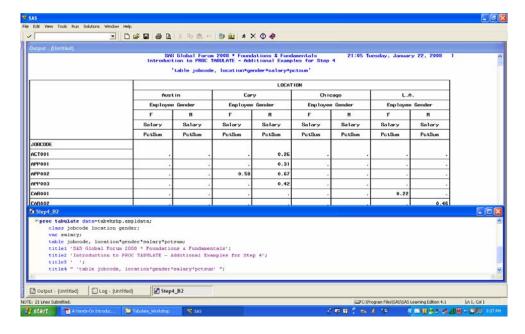
Here are a few more examples:

```
PROC TABULATE DATA=tabwkshp.empldata;
CLASS jobcode location gender;
VAR salary;
TABLE jobcode, location*gender*salary;
TABLE jobcode, location*gender*salary*PCTSUM;
TABLE jobcode, location*gender*salary*MEAN;
```

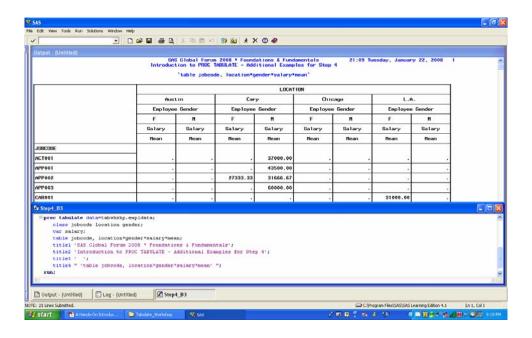
The first *TABLE* statement would generate a hierarchical breakdown of the sum of the salary amounts represented in each cell, according to values of *jobcode* and the nested values of *location* and *gender*.



The second *TABLE* statement would generate a hierarchical breakdown of the percentage of the total of salary amounts represented in each cell, according to values of *jobcode* and the nested values of *location* and *gender*.



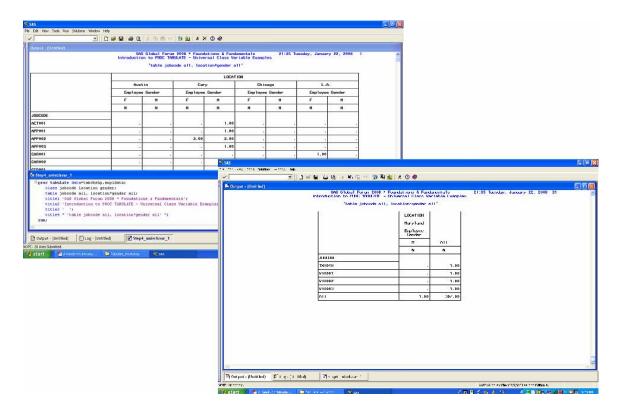
The third *TABLE* statement would generate a hierarchical breakdown of the average salary represented in each cell, according to values of *jobcode* and the nested values of *location* and *gender*.

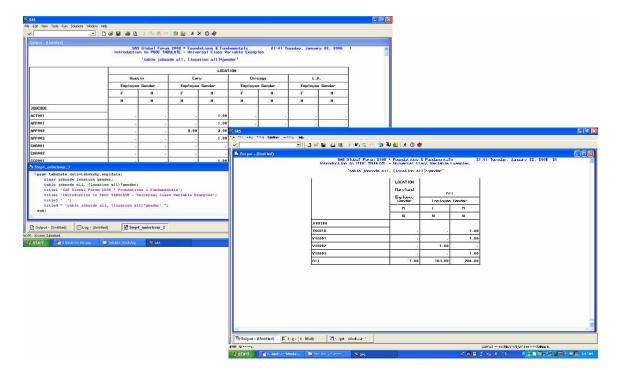


PROC TABULATE has a <u>universal class variable</u>, *ALL*, which can be used to generate totals for any specified class variable. Just concatenate the keyword *ALL* into the row or column expression of a *TABLE* statement.

```
PROC TABULATE DATA=tabwkshp.empldata;
CLASS jobcode location gender;
TABLE jobcode ALL, location*gender ALL;
TABLE jobcode ALL, (location ALL)*gender;
```

What is the difference between the tables produced by these two *TABLE* statements? Here are the results from the first *TABLE* statement:





And here are the results from the second TABLE statement:

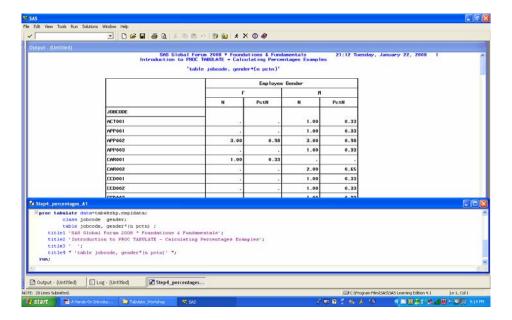
# CALCULATING PERCENTAGES WITH PROC TABULATE

PROC TABULATE has the capability of determining the percentage of the value in one cell to the value in another cell, or to the total of a group of cells, through the *PCTN* and *PCTSUM* statistics, respectively. Whenever you invoke PROC TABULATE to calculate percentages you, explicitly or implicitly, define the value that is to be used as the denominator. If you do not specify a particular denominator then, by default, PROC TABULATE uses the sum of the values in the *N* cells for the *PCTN* denominator, and the sum of the values in the *SUM* cells for the *PCTSUM* denominator.

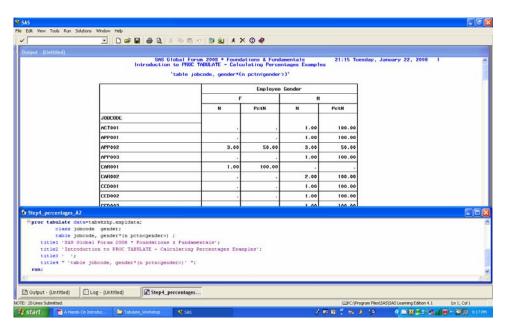
Brackets < > are used to explicitly specify the denominator that is to be used in the calculation of percentages.

```
PROC TABULATE DATA=tabwkshp.empldata;
CLASS jobcode gender;
TABLE jobcode, gender*(N PCTN);
TABLE jobcode, gender*(N PCTN
gender>);
```

The first *TABLE* statement specifies that the display should include the number of instances of values for *gender* occurring with each value of *jobcode*, and the percentages of those numbers to the total <u>across all</u> combinations of values of *jobcode* and *gender*.

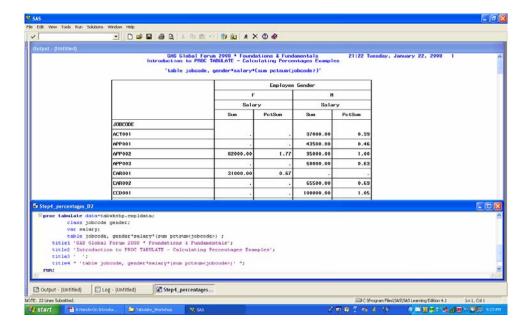


The second *TABLE* statement specifies that the display should include the number of occurrences of values for *gender*, and the percentage of that number to the total for *all values* of *gender* in each *jobcode* (that is, a row-percentage).



Here is another example of calculating percentages:

```
PROC TABULATE DATA=tabwkshp.empldata;
  CLASS jobcode gender;
  VAR salary;
  TABLE jobcode, gender*salary*(SUM PCTSUM<jobcode>);
```

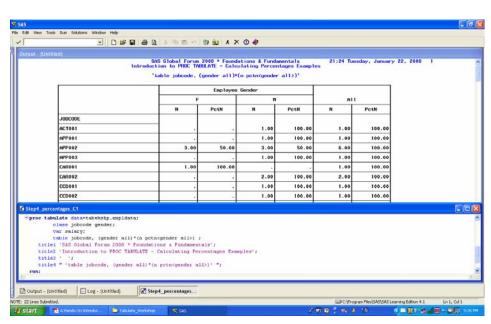


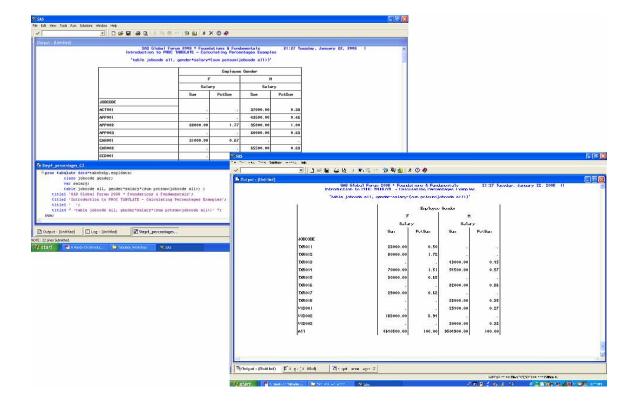
This *TABLE* statement specifies that the display should include breakdowns of the total *salary* amounts, and the associated percentages, for each combination of values of *jobcode* and *gender*, where the percentages are calculated in a column-wise manner.

Observe that, to obtain percentages by row, we use the column-expression in the "denominator definition"; and to obtain percentages by column, we use the row-expression in the "denominator definition."

And here are two more examples involving percentages, but these examples include the universal class variable, *ALL*.

```
PROC TABULATE DATA=tabwkshp.empldata;
  CLASS jobcode gender;
VAR salary;
TABLE jobcode, (gender ALL)*(N PCTN<gender ALL>);
TABLE jobcode ALL, gender*salary*(SUM PCTSUM<jobcode ALL>);
```





Notice that whenever row- or column-percentages are to be produced for a column- or row-expression that includes the *ALL* universal class variable, then *ALL* also must be included in the "denominator definition."

#### STEP 5 - LABELING AND FORMATTING THE TABLE

Now that we know how to define the basic structure of the tables we will generate, we would like to be able to make the tables more self-explanatory; that is, easier to read and interpret.

As in many other SAS procedures, you can use a *LABEL* statement to replace variable names with more descriptive headings for your class variables. There also is a way to specify temporary labels in a *TABLE* statement.

Similarly, *TITLE* and *FOOTNOTE* statements also can be used to enhance the tabular reports generated by PROC TABULATE.

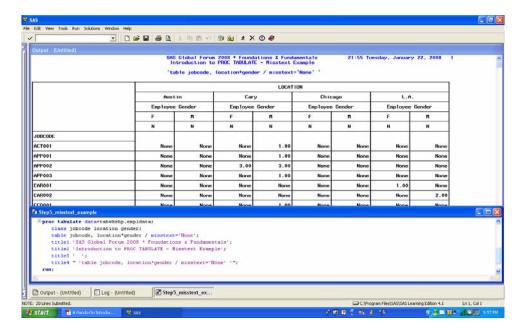
To assign labels to procedure-generated statistics and the universal class variable, we use the KEYLABEL statement.

```
KEYLABEL N = 'Count'
ALL = 'Total'
PCTN = 'Percent';
```

As in other SAS procedures, formats can be used to substitute labels for values of the classification variables. Formats also can be used to combine many values of the classification variables into a much smaller number of values to be printed in the report. We create custom formats by using PROC FORMAT, and we invoke those formats in PROC TABULATE either through a *FORMAT* statement, or by crossing *F=format-name*. in the *TABLE* statement with the particular variable.

The default for displaying cells with missing numeric values is a period. You can change the way missing values are displayed by using the *MISSTEXT*= option to define up to twenty characters of text that will print in the table cells whenever a particular combination of class variable values is not found in the input data set.

```
PROC TABULATE DATA=tabwkshp.empldata;
CLASS jobcode location gender;
TABLE jobcode, location*gender / MISSTEXT='None';
```



Here are a couple of useful *TABLE* statement options that can be used for customizing the appearance of tables:

- The RTSPACE= (or RTS=) option defines the total amount of space for the row headings. If there are several levels of headings for rows, then the space is divided equally among the levels, after subtracting the spaces that are needed for the vertical lines.
- Whenever a table produced by PROC TABULATE is too wide to fit on a single page, the procedure
  automatically splits the table, to span as many separate pages as are necessary for printing. For
  short, wide tables, the CONDENSE option could be specified on the TABLE statement, in order to
  print as many logical pages as possible on a single page, one below the other.

Some people think that traditional SAS output is ugly. Beginning with Version 7, the SAS System provided an ability to deliver procedure output in a flexible variety of file types and formats, through the SAS <u>Output Delivery System (ODS)</u>. Under SAS 9.1.3, ODS can be used to generate results as SAS data sets, output listings, PostScript, HTML, RTF, PDF, PCL, XML, Excel, and other output file types. ODS can be used to enhance tabular reports, by wrapping the PROC TABULATE code in ODS destinations, by changing fonts, colors, and other style attributes, and by adding graphics. For further information about ODS and TABULATE, consult the software documentation provided by SAS Institute.

Here is a final example, which illustrates several of the "Labeling & Formatting" techniques that I have described:

```
PROC FORMAT;

VALUE salfmt low-<12000 = 'Less than $12,000'

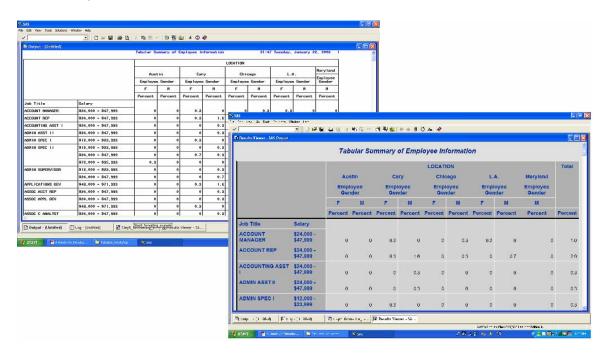
12000-<24000 = '$12,000 - $23,999'

24000-<48000 = '$24,000 - $47,999'

48000-<72000 = '$48,000 - $71,999'

72000-<96000 = '$72,000 - $95,999'
```

```
96000-<120000 = \$96,000 - $119,999'
                       120000-high = `$120,000 or more';
RUN;
ODS HTML BODY='tables.htm';
PROC TABULATE DATA=tabwkshp.empldata MISSING FORMAT=9.1;
     CLASS title location gender salary;
     FORMAT salary salfmt.;
     LABEL title = 'Job Title';
     KEYLABEL PCTN = 'Percent'
                                   ALL = 'Total';
     TABLE title*salary ALL, (location*gender ALL)*PCTN
                               / RTS=50 MISSTEXT='0';
     TITLE 'Tabular Summary of Employee Information';
RUN;
ODS HTML CLOSE;
RUN;
```



### **CONCLUSION**

PROC TABULATE is a very useful and very powerful procedure for constructing tabular reports containing descriptive statistical information, including hierarchical relationships among variables. It is well worth the necessary investment of time and effort for learning the intricacies and subtleties of its syntax. Concentrating on the five steps makes it much easier to learn how to code PROC TABULATE. We have only just "scratched the surface" of this wonderful procedure! But now you know enough to continue learning about it on your own. <u>Happy tabulating!</u>

### RESOURCES FOR FURTHER READING

Jeffery M. Abolafia & Stephen M. Noga, "The TABULATE Procedure: One Step Beyond the Final Chapter,"

- <u>SESUG '97 Proceedings</u> (1997), pp. 293-300; and
- Proceedings of the Twenty-Third Annual SAS Users Group International Conference (1998), pp. 839-844.

Jonas V. Bilenas, "Making Sense of PROC TABULATE (Updated for SAS9®)," <u>Proceedings of the SAS</u> Global Forum 2007, Paper #230-2007.

- Dan Bruns, "The Utter Simplicity of the TABULATE Procedure,"
  - <u>Proceedings of the Sixteenth Annual SAS Users Group International Conference</u> (1991), pp. 365-371; and
  - <u>Proceedings of the Twentieth Annual SAS Users Group International Conference</u> (1995), pp. 363-368; and
  - Proceedings of the Seventh Annual South-Central SAS Users' Conference (1997), pp. 54-60.
- Dan Bruns, "The Utter 'Simplicity?' of the TABULATE Procedure," <u>Proceedings of the Seventeenth Annual SAS Users Group International Conference</u> (1992), pp. 216-220.
- Dan Bruns, "Advanced Features of PROC TABULATE -- or -- The Utter Simplicity of the TABULATE Procedure The Sequel," <u>Proceedings of the Twenty-First Annual SAS Users Group International</u> Conference (1996), pp. 242-247.
- Dan Bruns, "The Utter 'Simplicity?' of the TABULATE Procedure -- The Final Chapter,"
  - Proceedings of the Twenty-Second Annual SAS Users Group International Conference (1997), pp. 251-256; and
  - Proceedings of the Seventh Annual South-Central SAS Users' Conference (1997), pp. 61-66;
  - <u>Proceedings of the Twenty-Ninth Annual SAS Users Group International Conference</u> (2004), Paper #241-29.
- Dan Bruns, "The Power and Simplicity of the TABULATE Procedure,"
  - Proceedings of the Twenty-Fourth Annual SAS Users Group International Conference (1999), Paper #152; and
  - <u>Proceedings of the Twenty-Fifth Annual SAS Users Group International Conference</u> (2000), Paper #152-25; and
  - Proceedings of the Twenty-Sixth Annual SAS Users Group International Conference (2001), Paper #148.
- Dan Bruns, "The Simplicity and Power of the TABULATE Procedure,"

  <u>Proceedings of the Twenty-Eighth Annual SAS Users Group International Conference</u> (2003),
  Paper #197.
- Dan Bruns & Ray Pass, "Battle of the Titans: REPORT vs. TABULATE," <u>Proceedings of the Twenty-Seventh Annual SAS Users Group International Conference</u> (2002), Paper #133-27.
- Dan Bruns & Ray Pass, "To REPORT or to TABULATE?: That is the Question!," <u>Proceedings of the Twenty-Ninth Annual SAS Users Group International Conference</u> (2004), Paper #122-29.
- Diane Louise Rhodes, "Speaking Klingon: A Translator's Guide to PROC TABULATE," <u>Proceedings of the Thirtieth Annual SAS Users Group International Conference</u> (2005), Paper #258-30.
- SAS Institute Inc., "The TABULATE Procedure," in <u>SAS® 9.1.3 Help and Documentation</u>, an HTML application that is installed as a component of the SAS System and is accessible from the SAS main menu by clicking "Help."
- SAS Institute Inc., "The TABULATE Procedure," Chapter 52 of <u>Base SAS® 9.1.3 Procedures Guide</u> (2006), available online at <a href="http://support.sas.com/documentation/onlinedoc/91pdf/sasdoc\_913/base\_proc\_8977\_new.pdf">http://support.sas.com/documentation/onlinedoc/91pdf/sasdoc\_913/base\_proc\_8977\_new.pdf</a>.
- SAS Institute Inc., <u>SAS Guide to TABULATE Processing, Second Edition</u> (Cary, NC: SAS Institute Inc., 1990).
- Bob Virgile, "The Right Approach to Learning PROC TABULATE," <u>SESUG '97 Proceedings</u> (1997), pp. 189-195.
- Tom Winn, "Introduction to Using PROC TABULATE,"
  - Proceedings of the Eighth Annual South-Central SAS Users' Conference (1998), pp. 316-326,

- Proceedings of the New Mexico SAS Users Conference (2002),
- Proceedings of the Arkansas SAS Day (2005), pp. 14-33,
- Proceedings of the 2006 Wisconsin-Illinois SAS Users Conference.

Tom Winn, "Advanced Features of PROC TABULATE,"

- <u>Proceedings of the Twenty-Fourth Annual SAS Users Group International Conference</u> (1999), Paper #153,
- Proceedings of the Ninth Annual South-Central SAS Users' Conference (1999), pp. 243-248.

Tom Winn, "A Hands-On Introduction to PROC TABULATE," <u>Proceedings of the SCSUG SAS Educational Forum 2007.</u>

## **CONTACT INFORMATION**

Thomas Winn, Ph.D. U.S. Department of Veterans Affairs DSS Support Office – Austin OI&T 1615 Woodward Street, 776 / 19F-4 Austin, TX 78772-0001

Telephone: 512 / 326-6180 E-Mail: Thomas.Winn@va.gov

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.