

PROC TABULATE: Getting Started

Art Carpenter
California Occidental Consultants

ABSTRACT

Although PROC TABULATE has been a part of Base SAS® for a very long time, this powerful analytical and reporting procedure is very under utilized. TABULATE is different; it's step statement structure is unlike any other procedure. Because the programmer who wishes to learn the procedure must essentially learn a new programming language, one with radically different statement structure than elsewhere within SAS, many do not make the effort.

The basic statements will be introduced, and more importantly the introduction will provide a strategy for learning the statement structure. The statement structure relies on building blocks that can be identified and learned individually and in concert with others. Learn how these building blocks form the structure of the statement, how they fit together, and how they are used to design and create the final report will be presented.

Once the foundation is laid a number of intermediate level techniques, options, and statements that provide the TABULATE programmer with a wide range of power and flexibility are presented. These techniques include those that are often underutilized even by the more experienced TABULATE programmer.

KEY WORDS

TABULATE, TABLE statement, CLASSLEV, KEYWORD, Style override option

INTRODUCTION

PROC TABULATE has its roots in a table generation program that had been developed by the Bureau of Labor Statistics (Rhodes, 2005). Much of the statement structure and syntax was directly adapted into PROC TABULATE.

The procedure primarily is used to generate data summary tables. Since TABULATE uses the same underlying statistical summarization engines as are used by REPORT and MEANS/SUMMARY, the same basic list of statistics are available in reports generated by TABULATE.

In addition to the PROC statement you will need to be familiar with three other primary statements:

- VAR list the analysis variables
- CLASS list the classification variables
- TABLES define the table analysis and structure

The VAR and CLASS statements are used in TABULATE in the same way as they are in MEANS/SUMMARY. And do not require any special knowledge to make use of them in TABULATE. They are, however, almost always required.

The TABLES statement is at the heart of the TABULATE procedure. It is this statement that defines the report table, what it contains, and what its structure will contain. It is the TABLE statement that requires a specific understanding in order to design and create a report.

TABLE STATEMENT - BASIC STRUCTURE

The TABLE statement (the keyword TABLE and TABLES are interchangeable) is used to define the structure of the report. There definitions that you might use:

- page definition used only when the report is to span multiple pages
- row definition defines the rows of a report within a page
- column definition defines columns of a report

Within the TABLE statement these three definitions are separated by commas. The statement may or may not contain a page definition (most do not). The row definition is also optional, however most TABLE statements will have a row definition. All TABLE statements will have a column definition.

```
table <page definition,> <row definition,> column definition;
```

The individual page, row, and column definitions are made up of building blocks and the building blocks are made up of items. The items can be analysis variables (VAR statement), classification variables (CLASS statement), formats, text, or other options. The building blocks themselves come in three forms:

- singular the item appears by itself table region, weight;
- nested two items are joined by an asterisk table region*sex, weight;
- concatenated two items are joined by a space table region sex, weight;

Creating a Singular Table

In the first TABLE statement associated with this example a single classification variable (REGION) is included in the row definition and

```
title1'Singular Table';
proc tabulate data=tabdat.clinics;
class region;
var wt;
table region,wt;
table wt,region;
run;
```

a single analysis variable (WT) is included in the column definition. The resulting table is singular in both the row and column definitions.

Exchanging the position of these two variables in the TABLE statement creates a table that has a horizontal orientation.

As is shown here, using two TABLE statements allows you to create multiple reports with a single PROC step. This can be very beneficial if the incoming data set is large as the data only has to be read into the procedure once.

Singular Table

	weight in pounds
	Sum
region	
1	780.00
10	1034.00
2	1078.00
3	1458.00
4	2228.00
5	1262.00
6	1980.00
7	604.00
8	640.00
9	1878.00

Example 1a

Singular Table

		region				
		1	10	2	3	4
weight in pounds	Sum	780.00	1034.00	1078.00	1458.00	2228.00
		region				
		5	6	7	8	9
weight in pounds	Sum	1262.00	1980.00	604.00	640.00	1878.00

Example 1b

In both cases the default statistic is the SUM. As was mentioned earlier we are not limited to this statistic. As a matter of fact most of the statistics that are available in MEANS/SUMMARY are also available in TABULATE.

Creating a Nested Table

Items are nested by separating them with an asterisk. In the first TABLE statement in this example, two classification variables are nested (SEX within RACE) to form the row definition.

```
proc tabulate data=tabdat.clinics;
  class race sex;
  var wt;
  table race*sex,wt;
  table race*wt,sex;
run;
```

Table of WT Nested within RACE

			patient sex	
			F	M
race				
1	weight in pounds	Sum	2413.00	5387.00
2	weight in pounds	Sum	1486.00	1508.00
3	weight in pounds	Sum	315.00	791.00
4	weight in pounds	Sum	454.00	.
5	weight in pounds	Sum	.	588.00

Example 2b

Table of SEX Nested within RACE

		weight in pounds
		Sum
race	patient sex	
1	F	2413.00
	M	5387.00
2	F	1486.00
	M	1508.00
3	F	315.00
	M	791.00
4	F	454.00
5	M	588.00

Example 2a

The asterisk is used to nest items so there is no reason why you cannot also nest analysis variables within classification variables. Either way the statistics represent the data from the interaction of RACE and SEX. The second TABLE statement (Example 2b) does this by nesting WT under RACE (in the row definition), while SEX appears as a singular item in the column definition.

The duplicated text that is associated with WT and the SUM statistic can thankfully be controlled, as is shown in a later example.

Creating Concatenated Tables

When a space, rather than an asterisk, is used to separate items in the TABLE

```
proc tabulate data=tabdat.clinics;
  class race sex;
  var wt;
  table sex race,wt;
run;
```

statement, the two items are concatenated. If these items are classification variables, a concatenated table is

generated. The concatenated items are stacked and their statistics are calculated independently.

In Example 3, the two classification variables SEX and RACE are separated by a space and the resulting table includes the independent results for both of these variables.

RACE Concatenated onto SEX

		weight in pounds
		Sum
patient sex		
F		4668.00
M		8274.00
race		
1		7800.00
2		2994.00
3		1106.00
4		454.00
5		588.00

Example 3

FORMING THE TABLE

The table is created by using combinations of singular, concatenated, and nested items. Often new users of the TABULATE procedure find it helpful to sketch out the general form of the table (rows and columns). From this sketch the first pass of the TABLE statement can be created. Further refinement of the table through formatting and the use of some of the other options shown below, can be applied on successive trials until the desired table is achieved. Most users quickly find that the 'trial and error' process shortens the learning curve substantially. Practice and soon you will find that you are creating tables that are very close to the desired outcome on the first try!

Selecting Statistics

TABULATE is very good at creating data summaries and univariate statistics. Any of the standard set of univariate statistics can be

```
proc tabulate data=tabdat.clinics;
class race sex;
var wt;
table race, sex*wt*median;
run;
```

generated from within TABULATE. These statistics include not only N, MEAN, and VARIANCE,

but also MEDIAN and several percentiles. These are generally specified by nesting them under the analysis variable to which they are to be applied. In this example the classification variables RACE and SEX are used to form the row and column definitions respectively. Notice that the analysis variable is nested within SEX, and that the MEDIAN statistic is tied to the analysis variable by nesting it to WT.

	patient sex	
	F	M
	weight in pounds	weight in pounds
	Median	Median
race		
1	162.00	185.00
2	155.00	179.50
3	105.00	105.00
4	113.50	.
5	.	147.00

Example 4

Using Parentheses to Form Groups

Concatenated items are joined using a space, and two or more concatenated items can be grouped by surrounding them with parentheses. Actually the group can contain any number or types of items including variables, statistics, formats, and options.

The following example contains a group of two analysis variables (WT and HT), which are nested under SEX, and a group of three statistics that are nested under the group of analysis variables. The nesting process creates a hierarchy in the table. For the column definition in this example SEX is listed first and is therefore at the top of the hierarchy.

```
proc tabulate data=tabdat.clinics;
class race sex ;
var wt ht;
table race, sex*(wt ht)*(n mean stderr);
run;
```

The two concatenated analysis variables are nested within SEX

which allows us to examine the statistics for these analysis variables for each of the values of SEX. Nesting is also used to associate the statistics with the analysis variables. Again the columns associated with the statistics will be nested under those of the analysis variables because in the TABLE statement,

	patient sex											
	F						M					
	weight in pounds			height in inches			weight in pounds			height in inches		
	N	Mean	StdErr	N	Mean	StdErr	N	Mean	StdErr	N	Mean	StdErr
race												
1	15	160.87	7.82	15	64.27	0.58	29	185.76	3.55	29	70.41	0.47
2	10	148.60	8.87	10	66.60	1.09	8	188.50	13.18	8	69.00	0.71
3	3	105.00	0.00	3	64.00	0.00	7	113.00	5.16	7	65.14	0.74
4	4	113.50	0.87	4	64.50	0.29
5	4	147.00	0.00	4	66.50	0.87

Example 5

the statistics are nested under the analysis variables. As was shown in an earlier example this nesting can be used on the row definition as well.

Using 'ALL' To Form Overall Statistics

The keyword ALL can be used to request statistics that are to be calculated across classification rows and columns. This gives us the ability to form group and subgroup statistics. The ALL keyword is used as if it was a classification variable and is concatenated onto the item over which you want to calculate the summary statistics.

```
proc tabulate data=tabdat.clinics;
class race sex ;
var wt ht;
table race all, (sex all)*ht*(n mean);
run;
```

In this example, ALL is concatenated to both the column and row classification variables. The table shows the summarizations for both these variables for the requested statistics.

	patient sex					
	F		M		All	
	height in inches		height in inches		height in inches	
	N	Mean	N	Mean	N	Mean
race						
1	15	64.27	29	70.41	44	68.32
2	10	66.60	8	69.00	18	67.67
3	3	64.00	7	65.14	10	64.80
4	4	64.50	.	.	4	64.50
5	.	.	4	66.50	4	66.50
All	32	65.00	48	69.08	80	67.45

Example 6

ASSIGNING CELL ATTRIBUTES

Although we have been able to generate useful summary tables using the few options and techniques already shown, the table is still not ready for final presentation. We will want to be able to control all aspects of the formation of the cells of the table, including text, formatting, cell size, labels, and much more. The following examples demonstrate a variety of these options, taken together they provide the necessary tools to generate tables that are ready to show the boss.

There are a number of options that were developed to control the table appearance for what is now known as the LISTING destination. Since most TABULATE users are now delivering their tables to other ODS destinations, these options will for the most part be ignored here.

Using Formats

You can use any of three primary ways to format the cells of a tabulate table. If a variable is already formatted or if you want to assign a format to a variable, the FORMAT statement can be used in TABULATE just as it is in virtually every other SAS procedure. You can also provide a default format for the numeric summarization cells on the PROC statement through the use of the FORMAT= option. Thirdly you can use an asterisk to associate (nest) a format with a statistic.

```
title1 'Assigning Formats';
proc format;
value $gender
'M' = 'Male'
'F' = 'Female';
run;
proc tabulate data=tabdat.clinics
format=8.3;
class sex;
var wt;
table sex all,wt*(n*f=2.0 mean stderr);
format sex $gender.;
```

	weight in pounds		
	N	Mean	StdErr
patient sex			
Female	32	145.875	5.730
Male	48	172.375	4.964
All	80	161.775	4.011

Example 7

In Example 7 a format has been created for use with the variable SEX. It is then assigned using the traditional FORMAT statement. The cells associated with the summary statistics (N MEAN MEDIAN) are given a default format (8.3) with the FORMAT= option to control the number of decimal places. Since N will be an integer, it should not have any decimal places, they are removed by associating the format 2.0 with that specific statistic.

Row Title Space

The term Row Title Space ,or RTS, refers to the area above the row headers. By default this area is left blank, however you can control its contents through the use of the BOX= option on the TABLE statement. Notice that this is a TABLE statement option, and as such, it follows a slash at the end of the table definitions.

```
proc tabulate data=tabdat.clinics
format=8.3;
class sex;
var wt;
table sex all,
      wt*(n*f=2.0 mean stderr)
      /box='Initial Conditions';
format sex $gender.;
run;
```

Using the Row Title Space

Initial Conditions	weight in pounds		
	N	Mean	StdErr
patient sex			
Female	32	145.875	5.730
Male	48	172.375	4.964
All	80	161.775	4.011

Example 8

You can also place the label of a variable in the RTS by using the variable name as the argument to the BOX option instead of the constant text. For this example we might have used the option /box = sex; and the label for the variable SEX (“patient sex”) would have appeared in the RTS.

Customizing Text

You can replace headings, labels, and other text by specifying it directly on the TABLE statement. Attach the text to the item of interest by using an equal sign followed by a quoted string. In this example we replace the word ALL with the string ‘Across Gender’.

```
proc tabulate data=tabdat.clinics;
class race sex;
var wt;
table race*wt*mean*f=5.1,
      (sex all='Across Gender');
run;
```

Controlling Text

race			patient sex		Across Gender
			F	M	
1	weight in pounds	Mean	160.9	185.8	177.3
2	weight in pounds	Mean	148.6	188.5	166.3
3	weight in pounds	Mean	105.0	113.0	110.6
4	weight in pounds	Mean	113.5	.	113.5
5	weight in pounds	Mean	.	147.0	147.0

Example 9

The text can also be attached to other items on the TABLE statement including variables, options, and statistics.

Removing Repeated Text

The ROW=FLOAT option can be used to collapse the row header space when a statistic or constant text is nested within row definitions. You probably noticed in Example 9 that the label and the statistic associated

```
proc tabulate data=tabdat.clinics;
  class race sex;
  var wt;
  table race*wt=' '*mean=' '
        ,sex
        /row=float box='Mean Weight';
run;
```

with the rows were repeated. This is, of course, very distracting. We can use the technique shown in the previous example to change the text to a blank, however the cells themselves will not go away. To remove the cells we use the ROW=FLOAT option. The FLOAT option removes columns that only contain blanks by dividing the RTS equally among non-blank columns.

Notice that the MEAN statistic is nested within WT even though WT has a text string associated with it.

Using FLOAT to Clear Row Labels

Mean Weight	patient sex	
	F	M
race		
1	160.87	185.76
2	148.60	188.50
3	105.00	113.00
4	113.50	.
5	.	147.00

Example 10

Replacing Missing Values

A table created by TABULATE, such as the one in Example 10, may contain missing values. These may or may not accurately reflect what you want to have displayed. In Example 10 there were no males of RACE=4 in the study, so a missing value is desirable, however if we were also showing the values of N, it would be correct to display these values as 0. We can force the replacement of missing values using the MISSTEXT= option.

```
proc tabulate data=tabdat.clinics;
  class race sex;
  var wt;
  table race
        ,sex*wt=' *(n=' ' mean=' ' )
        /box='Mean Weight'
        misstext='0';
run;
```

specified text string for each missing value in the table. Since it is applied to *all* missing values, this option is not very flexible. Fortunately, as is shown in the next example, user defined formats can be used to provide the needed flexibility.

```
proc format;
  value mzero
    . = '----'
    other = [6.2];
run;
```

In Example 11b the MZERO format is defined so that it can be applied to the MEAN values.

Missing values are mapped to a series of dashes and all other values are passed to the 6.2 format. The formatted value, '----' for missing values, will override the value supplied by the MISSTEXT option.

The MISSTEXT option inserts the

Filling Missing Values Using the MISSTEXT Option

Mean Weight	patient sex			
	F		M	
race				
1	15	160.87	29	185.76
2	10	148.60	8	188.50
3	3	105.00	7	113.00
4	4	113.50	0	0
5	0	0	4	147.00

Example 11a

```

proc tabulate data=tabdat.clinics;
class race sex;
var wt;
table race
      ,sex*wt=' '(n=' '
                        mean=' '*f=mzero.)
      /box='Mean Weight'
      misstext='0';

run;

```

The MISSTEXT= option has replaced the missing values for the N statistic, while the format MZERO. has been used for the MEAN statistic.

Filling Missing Values Using a Format

Mean Weight	patient sex			
	F		M	
race				
1	15	160.87	29	185.76
2	10	148.60	8	188.50
3	3	105.00	7	113.00
4	4	113.50	0	----
5	0	----	4	147.00

Example 11b

ORDERING THE HEADINGS

The ordering of the values of classification variables is based on the internal order of the unformatted values. This can be changed through the use of the ORDER= option. This option can take on the following values.

- internal order of the unformatted values (default)
- data order of the data itself
- formatted order based on the formatted values
- freq order based on the frequency

By default, the order is determined by the unformatted values. In the following step one of the two classification variables has been supplied with a format, however the format has not altered the order of the rows or columns.

```

proc format;
value $DIAGNOSIS
  '1'='Fracture'
  '2'='Flu'
  '3'='Ulcer'
  '4'='Laceration'
  '5'='Heart Disease'
  '6'='Cancer'
  '7'='Diabetes';

run;

proc tabulate data=tabdat.clinics;
class diag sex;
var wt;
table sex*wt=' '*n=' '
      ,diag
      /box='Patient Counts'
      row=float
      misstext='0';
format diag $diagnosis.;

run;

```

Controlling Order Defaults

Patient Counts	diagnosis code					
	Fracture	Flu	Ulcer	Heart Disease	Cancer	Diabetes
patient sex						
F	2	13	4	3	2	2
M	2	13	8	5	4	0

Example 12a

The values for the diagnosis code are in the order of the codes themselves (ORDER=INTERNAL). If we want to change the order to alphabetical based on the formatted value, the ORDER= option is set in the PROC statement. The following tables show the other alternatives.

When the ORDER= option is applied to the PROC statement it is applied equally to all of the classification variables. This can be a problem when we want the order to be determined differently for each of the classification

Controlling Order ORDER=DATA						
Patient Counts	diagnosis code					
	Flu	Cancer	Ulcer	Heart Disease	Diabetes	Fracture
patient sex						
M	13	4	8	5	0	2
F	13	2	4	3	2	2

Example 12b

Controlling Order ORDER=FORMATTED						
Patient Counts	diagnosis code					
	Cancer	Diabetes	Flu	Fracture	Heart Disease	Ulcer
patient sex						
F	2	2	13	2	3	4
M	4	0	13	2	5	8

Example 12c

variables. Fortunately we can now also apply the ORDER= option on the CLASS statement, as well as on the PROC statement. CLASS statement options follow a slash, which follows the list of one or more classification variables.

USING MULTIPLE CLASS STATEMENTS

The CLASS statement can be broken up into multiple statements and the individual statements can receive options. This increases our flexibility by applying specific options to specific classification variables. Some of these options mimic those that can be used on the PROC statement, while others are only used on the CLASS statement. Options available on the CLASS statement include:

- ASCENDING Control the order of values (usually ASCENDING is the default)
- DESCENDING Control the order of values
- EXCLUSIVE Limits values to those in the format
- PRELOADFMT Can be used to build missing classification groups from a format
- MISSING Missing values are valid levels (can also be used on the PROC statement)
- ORDER Define ordering criteria (can also be used on the PROC statement)

Controlling Order ORDER=FREQ						
Patient Counts	diagnosis code					
	Flu	Ulcer	Heart Disease	Cancer	Fracture	Diabetes
patient sex						
M	13	8	5	4	2	0
F	13	4	3	2	2	2

Example 12d

```
class diag sex / missing order=internal;
```

```
class diag / missing;
class sex / order=internal;
```

The two options on this CLASS statement will apply to both of the classification variables. However, this single CLASS statement could be rewritten as two separate CLASS statements, each with its own (and in this case different) options.

```
proc tabulate data=tabdat.clinics;
  class diag / order=formatted
            missing;
  class sex / order=freq;
  var wt;
  table (sex all)*wt=' '*n=' '
        ,diag
        /box='Patient Counts'
        row=float
        misstext='0';
  format diag $diagnosis.;
run;
```

In Example 13, two CLASS statements are used to assign options to the classification variables. The variable DIAG has two options and SEX has one.

Multiple CLASS Statements							
Patient Counts	diagnosis code						
	Cancer	Diabetes	Flu	Fracture	Heart Disease	Ulcer	Unkown
patient sex							
M	4	0	13	2	5	8	16
F	2	2	13	2	3	4	6
All	6	2	26	4	8	12	22

Example 13

USING CLASSDATA= AND EXCLUSIVE TO FILTER DATA

The CLASSDATA= option is used to identify a data set that contains levels of the classification variables that are to be included in the table. More than a fancy WHERE clause, the data set can also contain levels that are to appear in the report *even if they do not appear in the data!*

In this example the data set WORK.SELECTLEVELS has been created from a subset of the available levels of symptom (SYMP) and race (RACE). Additionally a level for each of these variables has been included which does not appear in the actual data.

```
data selectlevels(keep=race symp);
  set tabdat.clinics(where=(race in('1','4')
                             & symp in('01','02','03')));
  output selectlevels;
  if _n_=1 then do;
    race='0';
    symp='00';
    output selectlevels;
  end;
run;
```

This data set will be used as a filter by associating it with the CLASSDATA option on the PROC statement.

```
proc tabulate data=tabdat.clinics
              classdata=selectlevels;
  class race;
  class symp;
  var wt;
  table (race all)*wt=' '*n=' '
        ,symp all
        /box='Patient Counts'
        row=float
        misstext='0';
run;
```

Using the CLASSDATA Option
Without the EXCLUSIVE Option

Patient Counts	symptom code										All
	00	01	02	03	04	05	06	09	10		
race											
0	0	0	0	0	0	0	0	0	0	0	0
1	0	2	4	2	10	6	2	0	8		34
2	0	0	6	0	2	2	2	0	6		18
3	0	0	0	0	0	0	8	2	0		10
4	0	2	0	0	0	0	0	0	0		2
5	0	0	0	2	2	0	0	0	0		4
All	0	4	10	4	14	8	12	2	14		68

Example 14a

The resulting table has all the values from the two classification variables, and additionally the level for each variable that is not otherwise in the data. This technique can be used when you must have a level in the table that may or

may not

appear in the data.

Simply adding the EXCLUSIVE option to the PROC statement radically changes the table. The EXCLUSIVE option interacts with the CLASSDATA data set to give *only* those levels identified in the CLASSDATA data set.

```
proc tabulate data=tabdat.clinics
              classdata=selectlevels
              exclusive;
run;
```

Using the CLASSDATA Option
With the EXCLUSIVE Option

Patient Counts	symptom code				All
	00	01	02	03	
race					
0	0	0	0	0	0
1	0	2	4	2	8
4	0	2	0	0	2
All	0	4	4	2	10

Example 14b

You can also add or eliminate levels of the classification variables by using the preloaded format techniques described next.

PRELOADING FORMATS

In much the same way as the CLASSDATA option can be used as a super filter, preloaded formats can be used to control levels of classification variables. Within TABULATE, formats - especially user defined formats, can be used in several non-traditional ways, including data filters and report templates. The techniques are collectively known as “preloaded” or “preloading” formats. Although the implementation is somewhat different among different procedures, these techniques are available in several other procedures including MEANS, SUMMARY, and REPORT.

The first step in the process is to define a format for each classification variable that you wish to control. Levels can be included that do not exist in the data set (SYMP='00' and SEX=' '). The format \$SYMP. excludes values of SYMP ('03' - '10').

In TABULATE formats that are to be preloaded are identified on the CLASS statement using the PRELOADFMT option. The PRELOADFMT option will be ignored if one of the following options is not also present:

ORDER=DATA	option (on the CLASS statement)
ORDER=INTERNAL	option (on the CLASS statement)
PRINTMISS	option (on the TABLE statement)
EXCLUSIVE	option (on the CLASS statement).

```
proc format;
  value $symp
    '01' = 'Sleepiness'
    '02' = 'Coughing'
    '00' = 'Bad Code' ;
  value $gender
    'M', 'm' = 'Male'
    'F', 'f' = 'Female'
    ' ' = 'Unknown';
run;
```

We take advantage of the two formats defined above by specifying the PRELOADFMT option on both of the classification variables. Notice the use of the PRINTMISS option on the TABLE statement.

```
proc tabulate data=tabdat.clinics;
  class sex / preloadfmt;
  class symp / preloadfmt;
  var wt;
  table (sex all)*wt=' '*n=' '
    ,symp all
    /box='Patient Counts'
    row=float
    misstext='0' printmiss;
  format sex $gender. symp $symp.;
run;
```

Using Preloaded Formats
Without the EXCLUSIVE Option

Patient Counts	symptom code										All
	Bad Code	Sleepiness	Coughing	03	04	05	06	09	10		
patient sex											
Female	0	2	6	2	6	2	7	0	5	30	
Male	0	2	4	2	8	6	5	2	9	38	
All	0	4	10	4	14	8	12	2	14	68	

Example 15a

The resulting table shows the three formatted values of SYMP as well as the other unformatted values that appear in the data. Notice that there is also a column for SYMP='00' (Bad Code), which never appears in the data. Notice, however that although the same options are applied to SEX, the corresponding *extra* level ('Unknown') does not appear. This level does not show up because it was coded to a missing value, and by default the level associated with a missing value is excluded from the table. We can change this behavior with the MISSING option (Examples 13 and 15b).

The previous example has been slightly adjusted by adding the EXCLUSIVE option to both CLASS variables and the MISSING option to SEX so that the 'Unknown' row will be included.

```
proc tabulate data=tabdat.clinics;
  class sex / preloadfmt
             exclusive
             missing;
  class symp / preloadfmt
              exclusive;
  var wt;
  table (sex all)*wt=' '*n=' '
        ,symp all
        /box='Patient Counts'
        row=float
        misstext='0' printmiss;
  format sex $gender. symp $symp.;
run;
```

Using Preloaded Formats
With the EXCLUSIVE Option

Patient Counts	symptom code			All
	Bad Code	Sleepiness	Coughing	
patient sex				
Unknown	0	0	0	0
Female	0	2	6	8
Male	0	2	4	6
All	0	4	10	14

Example 15b

With the addition of the EXCLUSIVE option, levels of the classification variable that are not on the format are excluded from the table. The format is now acting like a template by both excluding and creating columns for the table.

USING THE STYLE OVERRIDE OPTION

PROC TABULATE accepts a STYLE= option in several of its statements. This option allows the user to control ODS attributes by replacing the attribute value supplied by the ODS style. The style override option can be applied to virtually any part of the table generated by the procedure.

The STYLE= Option

Styles can be applied to a number of areas within the table from general down to the specific cell.

	Table Area	STYLE= used on
❶	Box Cell	BOX= option
❷	Class Heading	CLASS statement
❸	Class Levels	CLASSLEV statement
❹	Analysis Variable Headings	VAR statement
❺	Statistics Headings (keywords)	KEYWORD statement
❻	Value Cells	PROC & TABLE statements
❼	Individual Cells	PROC & TABLE statements

```
title1 'TABULATE Using the Default Style';
proc tabulate data=tabdat.demog;
class sex;
var ht wt;
table sex*(ht wt), (n min median max);
run;
```

This simple TABULATE step was used to generate the following table (Example 16a) using the DEFAULT style and the HTML destination.

The callout numbers correspond to the items in the table above and to the locations on the report shown below (Example 16b).

TABULATE Using the Default Style

①	④ height in inches			
	N	Min	⑤ Median	Max
② patient sex				
F	32	62.00	⑥ 64.00	72.00
M ③	48	64.00	69.00	74.00

Example 16a

The attributes of each of these locations on the TABULATE output is determined by attributes that are defined by the style used by ODS. There are literally hundreds of these attributes associated with various parts of the table.

The STYLE= override option is used to change these attributes. There are too many attributes to cover in this short space, however a few that are commonly used are shown in the following table.

Controls	STYLE=	Possible Values
Font	font_face=	times, courier, (other fonts supported by the OS)
Text size	font_size=	6, 8 10 (sizes appropriate to the font)
Text style	font_style=	italic, roman
Text density	font_weight=	bold, medium
Text width	font_width=	narrow, wide
Foreground color	foreground=	color (color printers)
Background color	background=	color (color printers)

```
proc tabulate data=tabdat.clinics;
class sex / style={font_style=italic} ②;
var ht    / style={font_weight=bold ④
                font_size=8pt};
table sex, ht*(n min
               median*{style={background=white}} ⑥
               max)
        / box={label='Gender'
               style={font_face=helvetica}}; ①
run;
```

The availability of some values can be dependent on both the destination and operating system characteristics. The following code adds a number of override options to give you a sense of the effect of each one. Notice the use of the curly braces.

TABULATE Using the Style Override Option

Gender	height in inches			
	N	Min	Median	Max
patient sex				
F	32	62.00	64.00	72.00
M	48	64.00	69.00	74.00

Example 16b

- ① The font for the box text is set to helvetica.
- ② The label for SEX is italicized.
- ④ Two attributes are set in a single option.
- ⑥ The background color for this one statistic is set to white.

Using the KEYLABEL Statement

The KEYLABEL statement is used enhance the text associated with key words used in the TABLE statement.

These can include statistics and other key words such as ALL.

In this example all of the text associated with the keyword ALL is changed in one location using the KEYLABEL statement.

```
proc tabulate data=tabdat.clinics;
  class sex race;
  var ht;
  keylabel all='Total';
  table (race all)*(sex all),
        ht*(n mean);
run;
```

		height in inches	
		N	Mean
1	patient sex		
	F	15	64.27
	M	29	70.41
Total		44	68.32
2	patient sex		
	F	10	66.60
	M	8	69.00
Total		18	67.67

Using the CLASSLEV and KEYWORD Statements

The CLASSLEV and KEYWORD statements can be used to further refine the attributes associated with specific statistics (KEYWORD) or classification variables (CLASSLEV).

```
proc tabulate data=tabdat.clinics;
  class sex ;
  classlev sex/ style={font_style=italic
                      background=yellow};
  var ht;
  keyword n min max / style={background=white};
  table sex=' ', ht*(n min median max)
        / box='Gender';
run;
```

Using CLASSLEV and KEYWORD

Gender	height in inches			
	N	Min	Median	Max
<i>F</i>	32	62.00	64.00	72.00
<i>M</i>	48	64.00	69.00	74.00

Example 17

The CLASSLEV statement is used to set the attributes associated with the classification variable SEX. Like the CLASS statement the CLASSLEV statement can be broken up into multiple statements.

The KEYWORD statement assigns attributes to the *heading* associated with the selected statistics. Notice that the MEDIAN has been excluded from the list of statistics in this KEYWORD statement.

Assigning Attributes with Formats - Traffic Lighting

Attributes can be assigned explicitly as has been done in the previous examples or they can be assigned based on the values contained in the table.

```
proc format;
  value wtback
    low-100 = 'yellow'
    235-high= 'red';
  value wtfore
    235-high= 'white';
  value $genback
    'M' = 'cyan'
    'F' = 'pink';
run;

proc tabulate data=tabdat.clinics;
  class sex ;
  classlev sex/ style={background=$genback.};
  var wt;
  table sex=' ',
    wt*(n median
      (min max) * {style={background=wtback.
        foreground=wtfore.}})
    / box='Gender';
run;
```

This conditional assignment of attributes is extremely powerful, and it is implemented with user defined formats. In this example, foreground and background colors are assigned based on the values displayed in the table.

The STYLE= option on the CLASSLEV statement changes the value of the background color differentially for males and females.

The minimum and maximum

values in the data are used with the ranges in the WTBACK. and WTFORE. formats to set the colors for the foreground and background. Notice that the STYLE option has been nested under the statistics to which it is to be applied.

Creating Links

```
.... Code not shown....
proc format;
  value $genlnk ❶
    'M' = '#Males'
    'F' = '#Females';
run;

ods pdf anchor='Master';
ods proclabel='Overall';
proc tabulate data=tabdat.clinics;
  class sex ;
  classlev sex/ style={url=$genlnk. ❷
    foreground=blue};

  var wt;
  table sex=' ',
    wt*(n median min max)
    / box='Gender';
run;

ods pdf anchor='Males'; ❸
ods proclabel='Males'; ❹
title2 link='#Master' c=blue 'Males'; ❺
proc print data=tabdat.clinics;
  where sex='M'; ❻
  var lname fname ht wt;
run;
.... Code not shown....
```

Assigning Attributes with Formats

Gender	weight in pounds			
	N	Median	Min	Max
F	32	155.00	98.00	215.00
M	48	177.00	105.00	240.00

Example 18

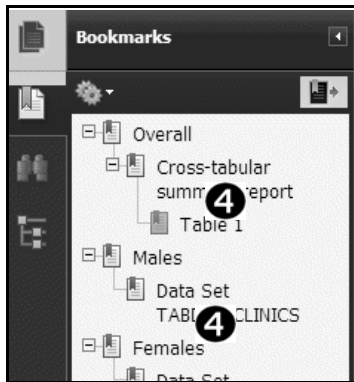
When creating electronic documents, we often want to be able to create links to other documents or to other locations within the current document. Within a TABULATE report it is possible to make virtually any set of characters a link by using the STYLE override option with the URL attribute. In the following example a PDF document is created that contains the output from a TABULATE and two PRINT procedure steps. The three reports are linked.

❶ A format is being used to assign the link location. The pound sign (#) identifies the link location as internal to the current file. Internal locations are specified with the ANCHOR= option ❸.

❷ The name of the file or, in this case, the internal location to which we will link is assigned using the URL attribute. Since this STYLE override option is on the CLASSLEV statement, the levels of the classification variables will be the links.

- ③ The ANCHOR= option marks an internal location in the current document to which we can link.
- ④ The ODS PROCLABEL statement changes how the results of this procedure are labeled in the PDF bookmarks.
- ⑤ The LINK= option can be used to create links in titles and footnotes.
- ⑥ A WHERE statement is used to create the listing for the males.

When using SAS9.2 under Windows, internal link locations and pointers for PDF documents should all be designated in lower case. In SAS9.1.3 mixed case is acceptable. The behavior may change in future releases of SAS9.2. The problem stems from the conversion of all the anchor locations to lower case when the document is rendered.



Example 19b ④

Gender	weight in pounds			
	N	Median	Min	Max
F	32	155.00	98.00	215.00
M	48	177.00	105.00	240.00

Example 19a ②

Obs	lname	fname	ht	wt
1	Smith	Mike	71	162
4	Marshall	Robert	67	155
6	Lawless	Henry	74	195

Example 19c ⑤

CALCULATING PERCENTAGES

Because we have to determine the appropriate denominator, calculating percentages can tend to be problematic. The denominator can be based on the entire report, a page, or more commonly on a specific row or column.

TABULATE supports several percentage generation statistics, which are tailored to determine the appropriate denominator.

Percentage applies to:	Percent Frequency (N)	Percent Total (SUM)
Report	reppctn	reppctsum
Page	pagepctn	pagepctsum
Column	colpctn	colpctsum
Row	rowpctn	rowpctsum

Column Percentages

Column percentages can be generated by using the COLPCTN and the COLPCTSUM statistics. The percentages

are based on the value in a given row using the denominator for the entire column. Although this example uses the ALL keyword, it is not required to use these statistics.

```
proc tabulate data=tabdat.clinics;
class race;
var wt;
table race all,wt*(n colpctn mean sum colpctsum);
run;
```

The resulting table has a column for both the percentage based on the frequency (COLPCTN) and the total value (COLPCTSUM).

		weight in pounds				
		N	ColPctN	Mean	Sum	ColPctSum
race						
1	44	55.00	177.27	7800.00	60.27	
2	18	22.50	166.33	2994.00	23.13	
3	10	12.50	110.60	1106.00	8.55	
4	4	5.00	113.50	454.00	3.51	
5	4	5.00	147.00	588.00	4.54	
All	80	100.00	161.78	12942.00	100.00	

In this particular example, summing the weights of all the patients for each level of RACE is a rather silly thing to do, but it does demonstrate the way that the COLPCTSUM value is calculated.

Example 20

Determining the Denominator

A specific denominator can be specified for a percentage statistic by using angle brackets to enclose the classification variable on which the denominator is to be based. In this example the PCTN statistic is used to calculate percentages based on three different denominators.

```
proc tabulate data=tabdat.clinics;
  class sex edu;
  table sex*(n='Patients' ❶
            pctn<edu>='% of row' ❷
            pctn<sex>='% of col' ❸
            pctn='Overall' ❹
            ),
        edu;
  format edu edulvl.;
run;
```

- ❶ The number of patients is displayed.
- ❷ Counting across levels of EDU, which forms columns, gives the row denominator.
- ❸ The denominator for the column percentage is based on counts across the two levels of SEX (rows).
- ❹ No denominator is specified so all observations are counted.

		Percentages Specified Denominator			
		years of education			
		High School	College	Grad. Sch.	
patient sex					
	F	Patients	8	16	8
		% of row	25.00	50.00	25.00
		% of col	25.00	47.06	57.14
	Overall	10.00	20.00	10.00	
M		Patients	24	18	6
		% of row	50.00	37.50	12.50
		% of col	75.00	52.94	42.86
		Overall	30.00	22.50	7.50

Example 21

SUMMARY

Although complex, the TABULATE procedure is powerful and flexible. Fortunately you do not have to 'know it all' to make use of its capabilities. Even mastering the few techniques described in this paper will allow you to create summarizations and reports that go far beyond the limitations of most other procedures.

Practice, experiment, try combinations of statements and options. There is a lot to learn, but there is so very much that you can accomplish.

ABOUT THE AUTHOR

Art Carpenter's publications list includes four books, and numerous papers and posters presented at SUGI, SAS Global Forum, and other user group conferences. Art has been using SAS® since 1977 and has served in various leadership positions in local, regional, national, and international user groups. He is a SAS Certified Advanced Professional Programmer, and through California Occidental Consultants he teaches SAS courses and provides contract SAS programming support nationwide.

AUTHOR CONTACT

Arthur L. Carpenter
 California Occidental Consultants
 10606 Ketch Circle
 Anchorage, AK 99515

(907) 865-9167
 art@caloxy.com
www.caloxy.com



REFERENCES

- Bruns, Dan, 2000, “The Power and Simplicity of the TABULATE Procedure”, *Proceedings of the Twenty-fifth Annual SAS® Users Group International Conference*, Cary, NC: SAS Institute Inc., 1624 pp.
- Chapman, Tasha, 2009, “How to Use PROC TABULATE”, *Proceedings of the Pacific Northwest Conference*, Cary, NC: SAS Institute Inc.,
http://www.sascommunity.org/mwiki/images/4/49/Proc_Tabulate_how_to_-_version_2.0.pdf
- Coleman, Ron, 1998, “The Building Blocks of PROC TABULATE”, *Proceedings of the Twenty-third Annual SAS® Users Group International Conference*, Cary, NC: SAS Institute Inc., 1660 pp.
- Haworth, Lauren, 1999, *PROC TABULATE by Example*, Cary, NC: SAS Institute Inc., 392 pp.
<http://www.sas.com/apps/pubscat/bookdetails.jsp?catid=1&pc=56514>
- Haworth, Lauren, 2000, “Anyone Can Learn PROC TABULATE, v2.0”, *Proceedings of the Twenty-fifth Annual SAS® Users Group International Conference*, Cary, NC: SAS Institute Inc., 1624 pp.
- Haworth, Lauren, 2000, “Elegant Tables: Dressing Up Your TABULATE Results”, *Proceedings of the 2000 Annual Conference of the Pacific Northwest SAS® Users Group*, Cary, NC: SAS Institute Inc., and in the *Proceedings of the 8th Annual Western Users of SAS® Software Conference*, Cary, NC: SAS Institute Inc.
- Rhodes, Diane Louise, 2005, “Speaking Klingon: A translator’s guide to PROC TABULATE”, *Proceedings of the Thirtieth Annual SAS® Users Group International Conference*, Cary, NC: SAS Institute Inc., paper 258-30.
<http://www2.sas.com/proceedings/sugi30/258-30.pdf>



TRADEMARK INFORMATION

SAS, SAS Certified Professional, SAS Certified Advanced Programmer, and all other SAS Institute Inc. product or service names are registered trademarks of SAS Institute, Inc. in the USA and other countries.
® indicates USA registration. Other brand and product names are trademarks of their respective companies.