Paper 085-2010

Tabular Reporting Data Standard

Frederick Cieri, FACtotum Data Consulting, LLC., Wynnewood, PA, USA

ABSTRACT

This paper demonstrates a reporting data standard to map a tabular output table to a single standardized data set. This process can significantly standardize program code, increase readability, help reduce validation work, help facilitate future revisions, and provide a data standard to build a reporting application. Due to different table output formats, report programs usually contain many different temporary data sets structures to convert the raw extracted data into a form usable to macros and procedures to create tabular output tables. This complicates future revisions and validation as different coding styles and temporary data set formats make reading program code from various programmers difficult. In most cases, the raw data comes from an administered database with formatting standards, but the data step transformations to produce report output differ from programmer to programmer. Summarized data sets can be transformed into a single standardized reporting data standard to be fed into a reporting application. Examples are given mapping three output tables to the reporting data standard termed the Table data set. SAS® code intended for those familiar with DATA steps, PROC SORT, and PROC TRANSPOSE is presented to show the ease in converting a Table data set to an output table format. A program diagram will be displayed to demonstrate how the tabular reporting standard can standardize a majority of the report code.

INTRODUCTION

In order to reduce the time to develop and maintain reports in a large programming group, a new design to standardize the code in a group of standard reports was needed. The task was difficult as the existing reports varied in style and syntax. Some reports were mostly hard coded while others used a large amount of macros. Some reports were done in house while other reports were outsourced. Department programming standards, standard operating procedures, and a large macro library were in place, but this did not solve the problem of standardizing the report code. Since a programming studio type application was not a possibility due to the cost, a solution using only BASE SAS was needed. The solution was to standardize the transaction data structures within a program by mapping the tabular output report to a data set. Standardizing the temporary data sets gave the benefits of a predictable data format, variable names, and variable types. The solution grew from the report application where the parameters and the input data were in databases. Using the same database framework on the temporary data sets within the report gave similar benefits of data uniformity, data edit checks, and utility creation. The tabular report output defined the tabular reporting data structure of the temporary data sets in a uniform way. Once the tabular reporting data structure was adopted then utilities were developed to aid with counts, percents, descriptive statistics, tabular reports, and many other derivations.

TABLE DATA SET CONCEPT AND EXAMPLES

To understand where to apply the Table data set process, let us look at the following generalized report table program flow with estimated percentages of the make up of each step.

1. (10%) Load parameters.
2. (20%) Extract and manipulate raw data .
3. (30%) Summarize data.
4. **(20%) Data set transformations in preparation for output table.**
5. (20%) Output table.

This paper concerns step 4 or the work involved to transform the summarized data based on the report requirements to a form ready for the output table code. Step 1 of load parameters consists of numerous options from the less complicated %let macro variables, macro parameters, and %window to the more complicated graphical user interfaces. Steps 2 of extraction and 3 of summarized data are difficult to standardize because of changing database structures and report requirements. Step 5 concerning the output table changes as well over time based on the format (doc, rtf, Word, etc.) of the output and the procedures (DATA _NULL_, PROC CONTENTS, ods, etc.) used to create the output. Spending the time to standardize step 4 can yield significant savings in resources and increase productivity because this code can more easily be reused as the database, requirements, and output procedures change.

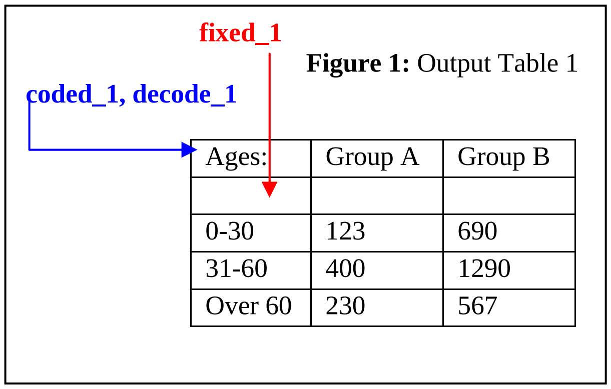


Figure 1: Output Table 1

Ages:	Group A	Group B
0-30	123	690
31-60	400	1290
Over 60	230	567

The following examples give an introductory sampling of the transformations needed to map a tabular output table to a standardized data set. Please look at Figure 1 with Output Table 1 with 3 columns. To transform this into a standardized data set, think of each cell in the body of the table as a multi-dimensional mapping of the header of 'Group A' and 'Group B' and the fixed column of 'Ages:'. The mapping in the body of the table of cell '123' would be ('0-30', 'Group A', '123'), and of cell '567' would be ('Over 60, 'Group B', '567').

To further demonstrate this idea, the Table data set for Output Table 1 below in figure 2 maps all nine entries in the body of the table into the cell column. The coordinate mapping system is expanded now to have numerical ordering columns row_order_1 to preserve the row ordering of the left most fixed column and coded_1 to preserve the header ordering. The cell coordinates are in the form of row order, fixed text, coded header order, decode header text, cell value. Now the cell '123' has the coordinates (2,'0-30', 1,'Group A','123') where 2 is a numerical row ordering value from variable row_order_1, '0-30' is the fixed term from variable fixed_1, 1 is the numerical header ordering value from variable coded_1, 'Group A' is the text to decode the header variable from variable decode_1, and '123' is the cell value in the cell variable. Note the columns 'Group A' and 'Group B' from Table 1 are now stacked upon each other. There are nine rows in the data set for the eight body cell entries and one entry for the 'Ages: ' header text above the fixed_1 column. The '_header_' term in the fixed_1 and cell column is to flag the entry for the header above the fixed column. To perform the standardization, the programmer is to write code to take the summarized data and make the Table data set for Output Table 1. Using data steps, macros and procedures, the Table data set would then be used to construct the Output Table.

Figure 2: Table Data set for Output Table 1

Obs.	row_order_1	fixed_1	coded_1	decode_1	cell
1	.	_header_	0	Ages:	_header_
2	1		1	Group A	
3	2	0-30	1	Group A	123
4	3	31-60	1	Group A	400
5	4	Over 60	1	Group A	230
6	1		2	Group B	
7	2	0-30	2	Group B	690
8	3	31-60	2	Group B	1290
9	4	Over 60	2	Group B	567

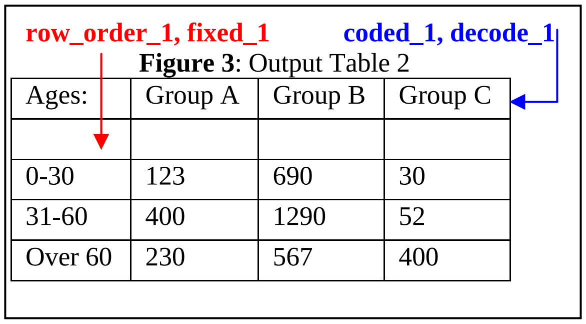


Figure 3: Output Table 2

Ages:	Group A	Group B	Group C
0-30	123	690	30
31-60	400	1290	52
Over 60	230	567	400

Expanding upon Table 1 in figure 1, figure 3 with Output Table 2 adds a 'Group C' column. The data set built from this table should now have 13 rows with 12 rows for the body of the table and one row for the header above the fixed column. The column 'Ages:' is termed fixed because with the output table design displayed the number of 'Group' columns are expected to change, but the 'Ages:' column is fixed to one column.

Figure 4: Table Data set for Output Table 2

Obs.	row_order_1	fixed_1	coded_1	decode_1	cell
1	.	_header_	0	Ages:	_header_
2	1		1	Group A	
3	2	0-30	1	Group A	123
4	3	31-60	1	Group A	400
5	4	Over 60	1	Group A	230
6	1		2	Group B	
7	2	0-30	2	Group B	690
8	3	31-60	2	Group B	1290
9	4	Over 60	2	Group B	567
10	1		3	Group C	
11	2	0-30	3	Group C	30
12	3	31-60	3	Group C	52
13	4	Over 60	3	Group C	400

In Figure 4, the Table data set for Output Table 2 has the same nine rows as in the data set for Output Table 1. Rows 10-13 reflect the additions due to the new ‘Group C’ column. By stacking entries with a vertical data set design, the Table data set design is flexible and can easily accommodate new columns added to the output table. This further demonstrates how a single standardized data set can map different tabular output tables.

Building upon Output Table 2 in figure 3, figure 5 with Output Table 3 shown below has a more complex three row header where each ‘Group’ has an overall count in header row 2 as well as counts and percents in header row 3. The data set mapped to this table will have 25 rows for the 24 entries in the body of the table and one row to capture the ‘Ages:’ header column. Output Tables 1 and 2 had a one row header with variables coded_1 and decode_1 to map the header. Since Output Table 3 has a three row header, three pairs of coded and decode variables displayed in blue below will be needed to map the header entries.

row_order_1, fixed_1

Figure 5: Output Table 3

coded_1, decode_1⟶

coded_2, decode_2⟶

coded_3, decode_3⟶

	Group A		Group B		Group C	
	(N=753)		(N=2547)		(N=482)	
Ages:	n	(%)	n	(%)	n	(%)
0-30	123	(16.3)	690	(27.1)	30	(6.2)
31-60	400	(53.1)	1290	(50.6)	52	(10.8)
Over 60	230	(30.5)	567	(22.3)	400	(83)

The Table data set for Output Table 3 is shown below in figure 6. The coded_1 decode_1 pair for header row 1 has the same values as the data sets for Output Tables 1 and 2. For header row 2, coded_2 always equals 1 because there is a one to one match with the ‘Group’ entries. For header row 3, coded_3 has values 1 and 2 to map the decode_3 values of ‘n’ and ‘%’. To create the column ordering to transform the Data set in Table 3 to the Output Table 3 table format, the trans_id column is derived by the programmer from the ascending order of the unique values of the coded variables. Please review the Output Table 3 data set below to further understand the mapping.

Figure 6: Table Data set for Output Table 3

Obs.	row_order_1	fixed_1	coded_1	decode_1	coded_2	decode_2	coded_3	decode_3	cell	trans_id
1	.	_header_	0		0		0	Ages:	_header_	1
	1		1	Group A	1	(N=753)	1	n		2
3	2	0-30	1	Group A	1	(N=753)	1	n	123	2
4	3	31-60	1	Group A	1	(N=753)	1	n	400	2
5	4	Over 60	1	Group A	1	(N=753)	1	n	230	2
6	1		1	Group A	1	(N=753)	2	(%)		3
7	2	0-30	1	Group A	1	(N=753)	2	(%)	(16.3)	3
8	3	31-60	1	Group A	1	(N=753)	2	(%)	(53.1)	3
9	4	Over 60	1	Group A	1	(N=753)	2	(%)	(30.5)	3
10	1		2	Group B	1	(N=2547)	1	n		4
11	2	0-30	2	Group B	1	(N=2547)	1	n	690	4
12	3	31-60	2	Group B	1	(N=2547)	1	n	1290	4
13	4	Over 60	2	Group B	1	(N=2547)	1	n	567	4
14	1		2	Group B	1	(N=2547)	2	(%)		5
15	2	0-30	2	Group B	1	(N=2547)	2	(%)	(27.1)	5
16	3	31-60	2	Group B	1	(N=2547)	2	(%)	(50.6)	5
17	4	Over 60	2	Group B	1	(N=2547)	2	(%)	(22.3)	5
18	1		3	Group C	1	(N=482)	1	n		6
19	2	0-30	3	Group C	1	(N=482)	1	n	30	6
20	3	31-60	3	Group C	1	(N=482)	1	n	52	6
21	4	Over 60	3	Group C	1	(N=482)	1	n	400	6
22	1		3	Group C	1	(N=482)	2	(%)		7
23	2	0-30	3	Group C	1	(N=482)	2	(%)	(6.2)	7
24	3	31-60	3	Group C	1	(N=482)	2	(%)	(10.8)	7
25	4	Over 60	3	Group C	1	(N=482)	2	(%)	(83)	7

SAS CODE TO CONVERT THE TABLE DATA SET

Since a single standardized data structure holds all of the information to map the three different tabular outputs above, the SAS code to transform the Table data set to a tabular output has a similar generalized format. The programming code and print outs to transform the Table data set for Output Table 3 to Output Table 3 are presented below.

```
%*** Transform the headers. ****;
 proc sort data=table_3 out=table_3_pre_header nodupkey;
   by  coded_1  coded_2 coded_3;
 run;
```

```
Proc print of Data= table_3_pre_header
Obs    coded_1    decode_1    coded_2    decode_2    coded_3    decode_3

   1        0                   0                    0       Ages:
   2        1      Group A      1       (753)        1       n
   3        1      Group A      1       (753)        2       (%)
   4        2      Group B      1       (2547)       1       n
   5        2      Group B      1       (2547)       2       (%)
   6        3      Group C      1       (482)        1       n
   7        3      Group C      1       (482)        2       (%)
```

```
Proc transpose data=table_3_pre_header out=table_3_header(drop=_name_)  prefix=col_;
   var decode_1 decode_2 decode_3;
run;
```

```
Proc print of data=table_3_header
Obs    col_1    col_2     col_3     col_4     col_5     col_6     col_7

 1           Group A   Group A   Group B   Group B   Group C   Group C
 2            (753)     (753)     (2547)    (2547)    (482)     (482)
 3     Ages:    n         (%)       n         (%)       n         (%)
```

```
%*** Transform the body. ****;
proc sort data=table_3 out=table_3_body_pre2;
 by  row_order_1 trans_id;
 where cell ne '_header_';
run;

Proc transpose data=table_3_body_pre2 out=table_3_body(drop=_name_) prefix=col_;
 by  row_order_1 fixed_1;
 var cell;
 id trans_id;
run;
```

```
Proc print of data=table_3_body
           row_
   Obs    order_1    fixed_1    col_2    col_3     col_4    col_5     col_6    col_7

    1       1
    2       2        0-30        123     (16.3)    690      (27.1)     30      (6.2)
    3       3        31-60       400     (53.1)    1290     (50.6)     52      (10.8)
    4       4        over 60     230     (30.5)    567      (22.3)     400     (83.0)
```

```
 %*** Combine the header and body ****;
 data  table_data_set_table_3;
  attrib row_order_1 length=8.
        col_1-col_7 length=$15.;
  set table_3_header
       table_3_body( rename=(fixed_1=col_1) )
      ;
  run;
```

```
Proc print of data=table_data_set_table_3
          row_
  Obs    order_1     col_1      col_2      col_3      col_4      col_5      col_6      col_7

   1        .                   Group A    Group A    Group B    Group B    Group C    Group C
   2        .                   (753)      (753)      (2547)     (2547)     (482)      (482)
   3        .        Ages:      n          (%)        n          (%)        n          (%)
   4        1
   5        2        0-30       123        (16.3)     690        (27.1)     30         (6.2)
   6        3        31-60      400        (53.1)     1290       (50.6)     52         (10.8)
   7        4        over 60    230        (30.5)     567        (22.3)     400        (83.0)
```

With the Table data set properly defined, the code above can be generalized into a macro to handle numerous dynamic cases for an unknown number of columns. For example, to transform the header, the key variables were coded_1, coded_2 and coded_3 for the proc sort and decode_1, decode_2 and decode_3 for the PROC TRANSPOSE. If a new table is to be created with a two row header then the table data set would have variables coded_1, coded_2, decode_1 and decode_2. The macro would have to figure out the variable names in the table data set with code using proc contents, data step functions, or sql dictionary. This derivation is not complicated because the prefixes of the variables are predefined to 'coded_' and 'decode_'. To end the derivation, the macro substitutes variables coded_1 and coded_2 for the proc sort and decode_1 and decode_2 for the proc transpose.

The three examples displayed are a starting point to demonstrate the Table data set approach. To map different table output formats, additional row ordering, fixed, coded and decode columns would be added. If fixed columns or header rows are added then the Table data set design expands horizontally. If new cell entries are added to the body of the table or the header then the Table data set design expands vertically. Additional columns for formatting such as alignment, formatting, page breaking, and other cases could also be required as well. The figure 7 example is displayed below with columns for alignment, formatting, and page breaks.

Figure 7: Table Data set for Output Table 2 with formatting and row added for pagebreaking

Obs.	row_order_1	fixed_1	coded_1	decode_1	cell	**align**	**format**	**pagebreak**
1	.	_header_	0	Ages:	_header_	L	$25	1
2	1		1	Group A		C	$12.	1
3	2	0-30	1	Group A	123	C	$12.	1
4	3	31-60	1	Group A	400	C	$12.	1
5	4	Over 60	1	Group A	230	C	$12.	1
6	1		2	Group B		C	$12.	1
7	2	0-30	2	Group B	690	C	$12.	1
8	3	31-60	2	Group B	1290	C	$12.	1
9	4	Over 60	2	Group B	567	C	$12.	1
10	1		3	Group C		C	$12.	1
11	2	0-30	3	Group C	30	C	$12.	1
12	3	31-60	3	Group C	52	C	$12.	1
13	4	Over 60	3	Group C	400	C	$12.	1
15	1		1	Group D		C	$12.	2
16	2	0-30	1	Group D	2	C	$12.	2
17	3	31-60	2	Group D	8	C	$12.	2

Additional work would also be needed on the transposed data set from the header and body as needed. All of these additions are noted as foreseeable work that needs to done by the programmer. Even with the additional columns, the core idea to define a single standardized Table data set structure to hold the tabular output is still maintained.

REPORT PROGRAM APPLICATION DIAGRAM

From the concepts section, the five steps of programming a report and are repeated below.

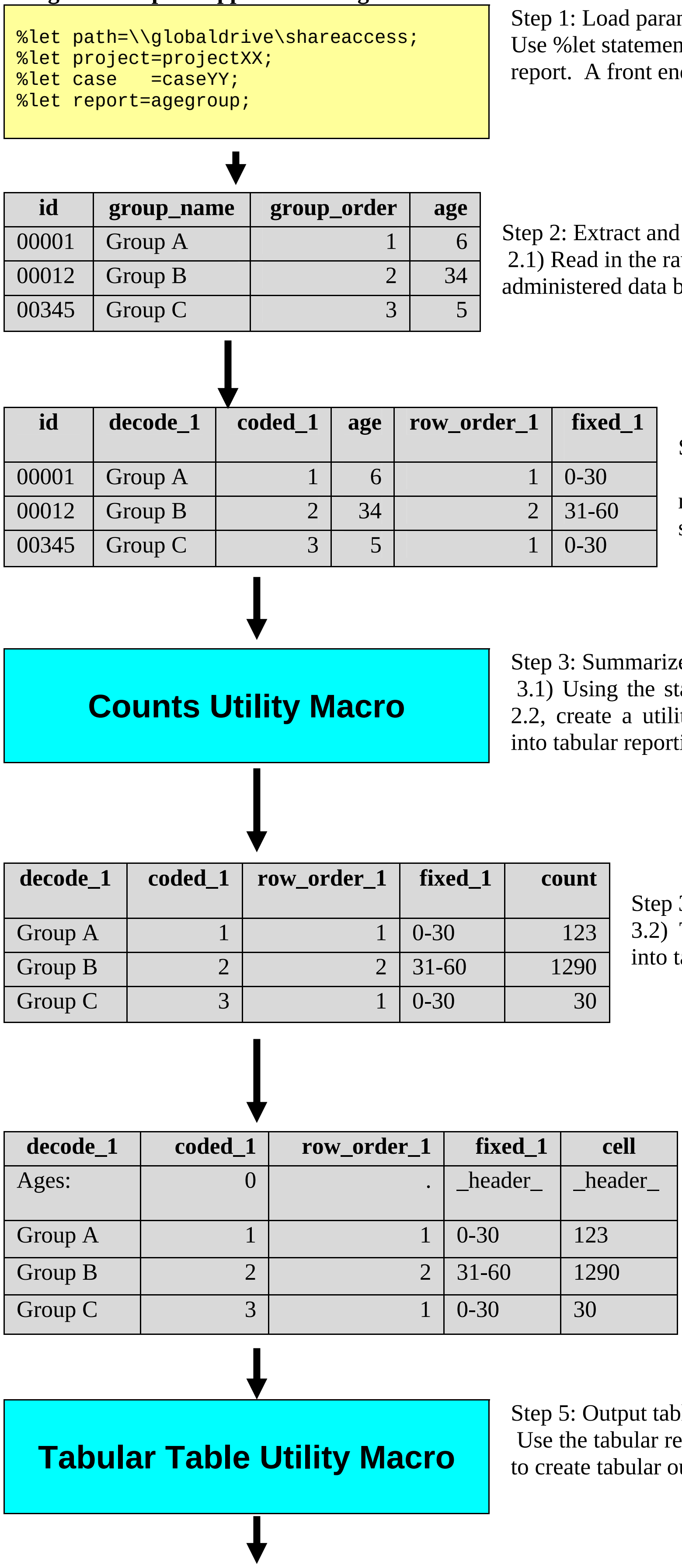
1. (10%) Load parameters.
2. (20%) Extract and manipulate raw data .
3. (30%) Summarize data.
4. **(20%) Data set transformations in preparation for output table.**
5. (20%) Output table.

Using the same steps, the standardization of the code can be maximized by using the tabular reporting data structure in step 4 to create utilities for steps 3 and 5. The new five steps of programming would now look like the following:

1. (10%) Load parameters.
2. (20%) Extract and manipulate raw data .
3. (5%) Summarize data.
4. **(60%) Data set transformations in preparation for output table.**
5. (5%) Output table.

Using the steps to maximize the use of the tabular reporting data structure, Figure 8 below is a report application diagram to explain the process to create the Output Table 2 example.

Figure 8: Report Application Diagram



```
%let path=\\globaldrive\shareaccess;
%let project=projectXX;
%let case   =caseYY;
%let report=agegroup;
```

Step 1: Load parameters
Use %let statements to initialize paths to the data and select a report. A front end user interface could also be used.

id	group_name	group_order	age
00001	Group A	1	6
00012	Group B	2	34
00345	Group C	3	5

Step 2: Extract and manipulate raw data
2.1) Read in the raw data. The data could be from an administered data base or adhoc input data sets.

id	decode_1	coded_1	age	row_order_1	fixed_1
00001	Group A	1	6	1	0-30
00012	Group B	2	34	2	31-60
00345	Group C	3	5	1	0-30

Step 2: Extract and manipulate raw data
2.2) Rename and create variables into tabular reporting data standard form. Data standardization starts at this point.

Counts Utility Macro

Step 3: Summarize data.
3.1) Using the standard variable names from the data in step 2.2, create a utility macro to perform counts and summarize into tabular reporting data standard form.

decode_1	coded_1	row_order_1	fixed_1	count
Group A	1	1	0-30	123
Group B	2	2	31-60	1290
Group C	3	1	0-30	30

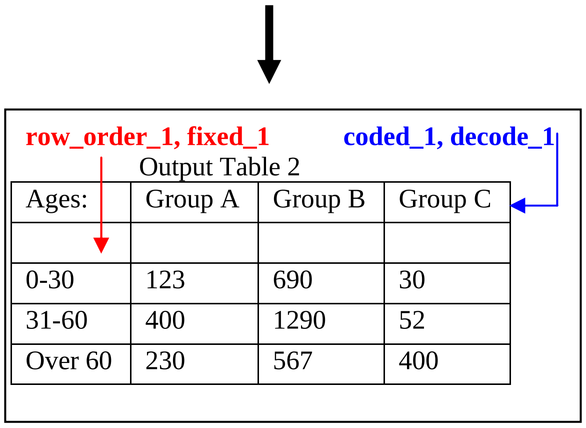
Step 3: Summarize data
3.2) The output of the macro would summarize into tabular reporting data standard form.

decode_1	coded_1	row_order_1	fixed_1	cell
Ages:	0	.	_header_	_header_
Group A	1	1	0-30	123
Group B	2	2	31-60	1290
Group C	3	1	0-30	30

Step 4: Data set transformations in preparation for output table
Derive the cell variable from count and add the header text above the fixed column in the row where cell='_header_'.

Tabular Table Utility Macro

Step 5: Output table.
Use the tabular reporting data standard to create a utility macro to create tabular output.



Ages:	Group A	Group B	Group C
0-30	123	690	30
31-60	400	1290	52
Over 60	230	567	400

Step 5: Output table
Final Output.

Instead of starting the standardization at step 4, the diagram above shows data standardization starting in step 2 and going through step 5. This approach maximizes the use of the tabular reporting data standard. With proper planning and design, the tabular reporting data standard could encompass 60% to 80% of the program code.

BENEFITS

After programmers get beyond the learning curve of understanding the Table data set design, enhanced productivity should be attained. Due to the single standardized data set design of the Table data set, the programming code and data sets created should be more readable in the data transformation section prior to creating the output table. The Table data set design can be rigorously documented leading to improved training materials. The Table data set design is standardized which allows for fool proofing, edit checks, and data integrity checks. This means variable names and types can be checked as well as the overall data structure. For example, the first row header text variable or decode_1 must be present and must be character. Highly reusable macros can be developed to assist the programmer in creating the Table data set transformations and in performing fool proof or edit checks. Likewise, global macros can be developed to take the Table data set and create a data set or sets needed for the output table section. As the output table section or step 5 of the previously presented program flow changes to handle output table file format changes (doc, rtf, html, etc.), macro changes, or procedure changes, the Table data set section or step 4 can for the most part stay the same. Since the Table data set is standardized with a predictable uniform format, only the global macro that transforms the Table data set into a form readable to output table macros and procedures needs to change. This saves time because instead of changing every report program only a global macro needs to be updated. Implementing the tabular reporting data standard gives the benefit of programmers speaking the same language when developing reports. For example, if two fixed columns are to be present in a report then the character variables fixed_1 and fixed_2 must be present in the table data set. If a programmer views a Table data set with the variable decode_4 then this means a four row header is present in the output table with decode_4 holding the text for the fourth row of the header.

CONCLUSION

In three examples, this paper outlined a tabular reporting data standard to map a tabular output table into a data set. This brings order and standardization to a signification amount of the data sets and code used to prepare the tabular output table. Highly uniform code should improve validation efforts because reviewers will not have to understand many different programming styles. Improving the readability of a report program, the Table data set design gives the programmer a definable endpoint for the format of the data set prior to the output table code. The design emphasized a highly viewable and uniform data set structure as opposed to heavy macro code. Leading to even more standardization and productivity gains, the Table data set structure can be further extended to earlier steps of the report program such as summarizing the extracted data into descriptive statistics, counts, and percentages. In order to meet more advanced formatting needs, add additional columns to the design as needed to meet your programming requirements.

ACKNOWLEDGEMENTS

The author would like to thank the following people who tested, vetted, and gave feedback on the Tabular Reporting Data Standard. Barry Cohen, Girishanthy Krishnarajah, Mark Tumelty, Carl Herremans, Mary Varughese, Anil Golla, Mei Dey, Jacek Krolikowski, and Eunice Ndungu.

CONTACT INFORMATION

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

Frederick Cieri
FACtotum Data Consulting, LLC
SAS Consultant, MS Statistics
304 Hathaway Lane
Wynnewood, PA 19096-1905
USA
610-658-2940
fecieri@yahoo.com

Please send an email to the author or go to sasCommunity.org in order to receive a detailed SAS program for the Output Table 3 example.