

Producing Standard Tables with a Plethora of Options: A Truly Data-Driven Approach

Leslie T. McBrayer, The Procter & Gamble Co.

ABSTRACT

In any situation where similar tables are produced on a regular basis, automation of table generation is in order. However, as many programmers can attest, two tables which look similar may in fact require vastly different amounts of processing to produce the results. Some differences between outputs are easily identifiable, such as the titles or which variables to include; other differences are more subtle, such as deleting certain observations based upon varying criteria. These "hidden" options which would make a table generating system more flexible also make its implementation more challenging.

The Data Table Generation (DTG) System, currently under development at Procter & Gamble, handles this challenge by building upon the concept that it is easier to manipulate data than code. The system does not send a list of parameters directly to a routine which contains extensive conditional code; instead, it combines data containing the options users select with data about the tables to be produced. This final data set is converted into SAS code, which is then executed to produce the tables. This paper describes the process and the supporting data structures used by the DTG System.

PROJECT BACKGROUND

In 1993, a team at Procter & Gamble Pharmaceuticals evaluated the processes used in writing clinical reports, which are used in submissions to the FDA for new drug approval. This team observed that different reports used similar tables to summarize the data, but each table for each study was typically generated by a different program, each written by a different programmer. Similarity between tables was not always utilized. Therefore, they recommended that the company: (1) create standard tables, which could be used across reports, and (2)

create a software system to generate those tables in a predictable, validated fashion.

Standard tables provided a solid starting place for the design of a reporting system, specifying the precise choices a user could make concerning the basic characteristics of each table. Typical options for a summary table included titles, footnotes, variables to report, and whether to use the raw values or, say, change from baseline. Above and beyond the basic characteristics of a table, however, are more subtle differences in data or reporting across projects. For example, the team which created the standard tables identified 37 different methods to subset the data, and requested that the system allow users to specify any combination of those methods.

A challenge in designing the system was to enforce the standards *and* provide this level of flexibility *and* make the system maintainable — so when a 38th method for subsetting was identified, for example, it could be easily incorporated into the existing system.

THE NEED FOR DATA MANIPULATION

The first step in the design of the system was to analyze how tables are produced in the most general case. The basic steps are fairly straightforward:

- (1) Obtain an existing data file.
- (2) Perform desired manipulation on the data to prepare for summarization or listing.
- (3) Summarize the data to meet the needs of the table.
- (4) Print the results.

Since the DTG System was to produce standard tables, parts (3) and (4) were fixed; the system also had access to standardized data sets, so part (1) was fixed. However, neither the data standards nor the table standards could completely remove the need for step (2).

Analysis of existing programs written by statistical personnel at Procter & Gamble yielded the following set of data manipulation functions typically performed prior to table production:

- attach labels or formats
- specify the order in which values of a variable should appear (e.g., Placebo should appear before Active)
- specify how to handle repeated measurements (e.g., use only the first, or last, or some combination such as the mean)
- subset the data

Obviously, this is not a comprehensive list. However, it covers a large portion of the needs of the initial system users, and is a finite, achievable list of options for implementation in the initial version of the system. It is anticipated that this list will grow over time.

To simplify system design, all data manipulation could have been included within the code to produce each table. However, from a maintenance perspective, this was unwise, as the set of manipulations was expected to change over time, and in this case the changes would have to be made in multiple places within the system.

AVOIDING REPETITION OF STEPS

For some tables, users may want to perform very specific data manipulations; e.g., one particular summary may be run against a special subset. However, there are many cases where the same rules or functions should be applied across a number of tables. For example, it would be natural for a certain variable to have the same format and label on all tables which include that variable.

When programmers face the situation of manipulating data the same way across tables, they will often create several tables via the same program, processing the input data set only once. If an automated system emulates this data flow and code organization, the number of times a user has to specify the same data manipulation steps is potentially reduced, and the processing efficiency is certainly increased. However, this method requires that the data manipulation portion of

the code be independent of the code to produce the tables.

Therefore, for maintenance and efficiency reasons, the DTG System separates the specifications for data manipulation (wherein lies the plethora of options) from the specifications for standard table production.

COLLECTING AND SENDING INPUTS

Given the relatively generic nature of the available data manipulation steps, a user could specify the same step multiple times even within a single table – e.g., multiple subsets, or multiple variables to format – and could specify a variety of combinations of steps. Conceptually, the data manipulation steps were a set of building blocks which could be put together in a wide variety of ways. Thus, the number of inputs the user could supply could vary dramatically, even for two versions of the same standard table.

However, the inputs could be logically grouped according to whether they were associated with a table or a particular data manipulation step. In the user interface, users had to first specify what they wanted to do before the system could request additional information about specific inputs required to support that action. When the interface sent the information to be processed, it could include contextual information so that the processing portion of the system knew precisely which tables and steps were desired.

In the DTG System, inputs are collected via a GUI on the PC and then uploaded to a VAX for batch processing. The primary file created on the PC which contains user-supplied input is called the user selections file.

This user selections file is a SAS data set which includes fields for the table name (template), the name of the data manipulation step (function), the request, and the input provided by the user (response). For example, a demographics listing, named demlis01, may require three inputs from the user: the title, footnote, and variable list. If the user specifies a title of "Table 1," a footnote of "kg: kilograms," and variables age, height, and weight, then the following records are

included in the user selections file (response is truncated to match available space):

Template	Function	Request	Response
demlis01	.	title	Table 1
demlis01	.	footnote	kg: kilo
demlis01	.	varlist	age
demlis01	.	varlist	height
demlis01	.	varlist	weight

One of the choices for function is "s_subset," a simple subset. Its inputs, or requests, include the name of the variable to subset on, the operator to use (e.g., "in" or "not in"), and the values to consider. If a user specifies that demlis01 should use only male subjects, for example, records such as those shown below would be added to the user selections file:

Template	Function	Request	Response
demlis01	s_subset	var	sex
demlis01	s_subset	operator	in
demlis01	s_subset	value	M

The user selections file may also indicate whether a function should be applied across tables. The keyword "_all_" is used to indicate, for example, that a function should be applied to all tables. Continuing the scenario above, if a user indicates that for all tables, the treatment variable should be labeled "Dose (mg)," then the user selections file would contain records similar to those shown below:

Template	Function	Request	Response
all	newlabel	var	trtgroup
all	newlabel	label	Dose (mg)

INTERPRETING THE INPUT

At the heart of the DTG System is a macro library which contains the basic building blocks for creating a table or manipulating the data. Each data manipulation step, for example, is performed by a macro whose name may be reflected in the "function" field of the user selections file. The input parameters for each macro correspond to the inputs collected from the user. For example, the definition for s_subset appears as:

```
%macro s_subset(var=, operator=, value=);
...
%mend s_subset;
```

This close relationship between functions, inputs, and available macros provides a

relatively simple translation from the user selections file to the code which must be submitted to create a table. In the above example, the user selections file:

Template	Function	Request	Response
all	newlabel	var	trtgroup
all	newlabel	label	Dose (mg)
demlis01	s_subset	var	sex
demlis01	s_subset	operator	in
demlis01	s_subset	value	M
demlis01	.	title	Table 1
demlis01	.	footnote	kg: kilogram
demlis01	.	varlist	age
demlis01	.	varlist	height
demlis01	.	varlist	weight

would be used to create code similar to:

```
%newlabel( var = trtgroup
            label = Dose (mg));

%s_subset( var = sex,
            operator = in,
            value = M);

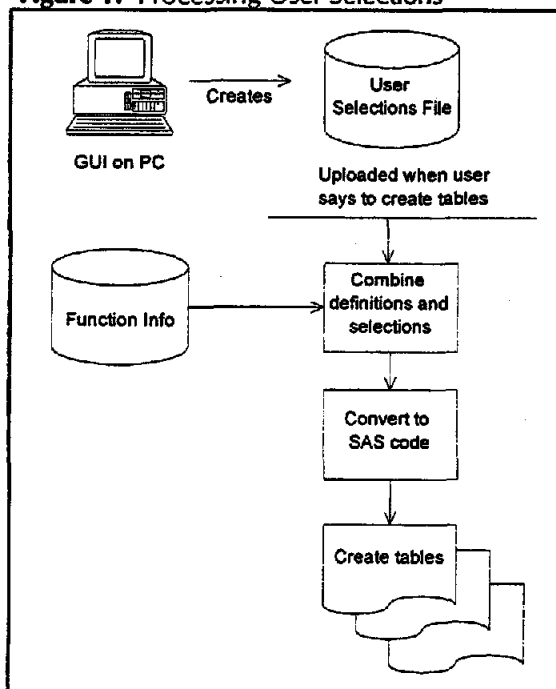
%demlis01( title = Table 1,
            footnote = kg: kilograms,
            varlist = age height weight);
```

The routine which converts the data set to code includes a single data _null_ step, with put statements to write out each line of code. Logic to determine the placement of the special characters relies upon the first *variable* and last *variable* data step construct. Note that the user selections file contains additional fields not included in the above example to assist in this conversion. For example, since the same table name could have been selected more than once, there is a numeric count field associated with the table name to distinguish between separate instances.

Notice that there is a single input parameter to table demlis01 for the title (and footnote) rather than a set of title1 ... titlen parameters. Yet more than one title line is often required, and the number of title lines may vary. Multiple title lines are supported by the code for demlis01, which expects that each line will be separated by a backslash (\). Also, the assigning of a label and the subsetting of the data are both events which could take place within a single data step, rather than requiring that %newlabel and %s_subset each contain a separate data step.

In order to handle these special requirements, the DTG system stores information about the functions and tables available, such as special parsing requirements or whether the function includes its own data step. This information, called the function definition data set, is joined with the user selections file prior to converting the file into code (Figure 1).

Figure 1: Processing User Selections



Function definitions for %newlabel and %demlis01 include information similar to the following:

Function	DataStep	Request	Delimiter
newlabel	no	var	
newlabel	no	label	
demlis01	yes	title	\
demlis01	yes	footnote	\
demlis01	yes	varlist	

Thus, the rules for generating code from the user selections file may be slightly enhanced. For example, if the user specifies an additional title such as "Demographic Listing," the above user selections file, combined with function definitions, could translate to the following code:

```

data new; set old;
  %newlabel( var = trtgroup
             label = Dose (mg));

  %s_subset( var = sex,
             operator = in,
             value = M);
run;

%demlis01( title = Table 1 \
           Demographic Listing,
           footnote = kg: kilograms,
           varlist = age height weight);
  
```

ADDITIONAL CONSIDERATIONS

Obviously, the code fragment shown above, containing only macro invocations, would not be sufficient to produce a table. At a minimum, the generated code must include libname statements to reference the existing data used to create the table, as well as the macro library. These lines of code are generated at the beginning of the data _null_ step.

In addition, note that part of the system design was to emulate programs written manually which process the input data once, then create multiple tables using the same program, to reduce duplicate processing. The system also had to allow users to specify data manipulation for a particular table. Therefore, careful attention was paid to the flow of data within the generated programs.

Within each macro, it was assumed that the primary input data set would be the last one used; SAS keeps track of this and may be referenced using the keyword "_last_". The data set to consider may also be set using the SAS statement:

```
options _last_ = data set name;
```

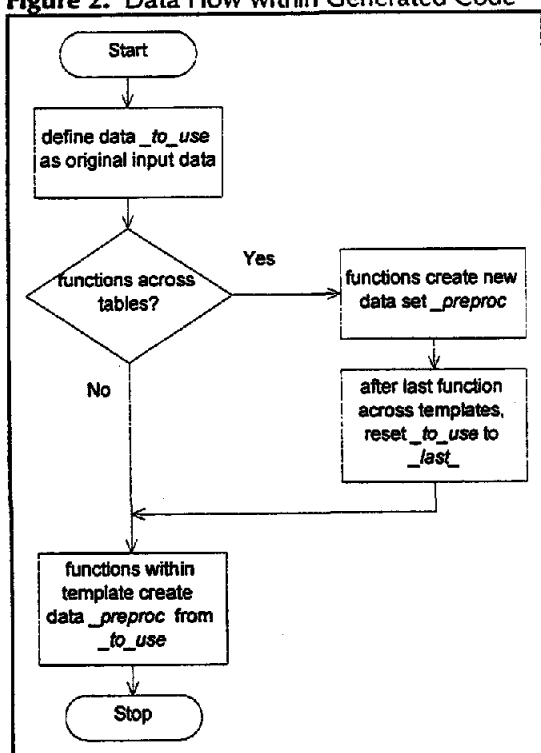
The first step in the generated code, therefore, after inserting a program header and establishing librefs and options, was to store the data to be processed in a work data set called "_to_use." Any data manipulation to be performed across all the tables produced by the current generated program was included first; once this was completed, code was included to store the result back into "_to_use."

Each data manipulation macro was designed so that the output data set was always a work data set called "_preproc," to distinguish the manipulated data from the data shared across tables.

Thus, after common data manipulation was included in the generated code, each table and its associated data manipulation steps could be included sequentially. Each table started with data set "_to_use" as defined above, and if any data manipulation was required, it produced a separate data set "_preproc." Thus, "_to_use" was intact and available for any subsequent tables.

A high-level flow diagram of this process is shown in Figure 2. Sample code similar to that generated by the DTG System is shown in Attachment 1.

Figure 2: Data Flow within Generated Code



CONCLUSION

Creating code from a data set provides the DTG system the flexibility to combine a number of steps in a variety of ways, tailored specifically to user selections. In addition, the code which makes up the core of the system – the conversion routine – is static, and does not

need to change when tables are changed or data manipulation functions are added. Validation and maintenance are both made much easier by this approach.

The primary cost of this method is additional overhead in processing time. This cost was deemed acceptable in the case of the DTG System since the system is designed to generate a large number of tables in a batch process, and the conversion process takes less time than the creation of a single table.

To date, users have been enthusiastic about the system. The time to make a few selections and have the system convert them to code is far less than the time required to create and validate new code. Users have a chance to spend more time on the interesting new analyses and reports rather than dwelling upon the creation of standard tables.

SAS is a registered trademark of SAS Institute Inc. in the USA and other countries.

Attachment 1: Sample of Generated Code (edited version)

```

/*****
This program was generated by the Data Table Generation (DTG) System.

File name: DEM.SAS
File location: SYS$U9:[DTG_DEV.study.TEST.123456.LTM01JUN]
DTG System user: GUEST
Date file created: 01JUN95
Protocol(s): TEST 123456
Purpose: Create DEM-type standard table(s)
Pre/Post Conditions: Clinical data extracted by DTG System, and
                      formats specified in DTG user interface, are
                      available only during system execution.
                      Stored, compiled macros are always accessible.
Data/files accessed: See libname statements at top of code.
Permanent files created: None.
*****/

libname MACROS 'SYS$U9:[DTG_DEV.system.runpgms.macros]';
libname LIBRARY 'SYS$U9:[DTG_DEV.study.TEST.123456.LTM01JUN]';
libname DATA 'SYS$U9:[DTG_DEV.study.TEST.123456.LTM01JUN]';

options mstored sasmstore=macros ps=50;

data _to_use; set DATA.DEM; run;

%nlblfmt (nfmt = $SEQFMT.,
         nlabel = Sequence,
         varnm = TrtGrp)

data _to_use; set _last_; run;

%ssubset (operator = not in,
         value = 35900901 35900911,
         var = PatNo)

%demlis02 (titles = Anthropometric Data and Vital Signs: Evaluable Subjects,
         footnote = Subjects 35900901 and 35900911 excluded,
         varlist = Age Hgtno Wgtno Tmpno RsprRt HrtRtno SystBPno DiasBPno)

options _last_ = _to_use;

%demlis02 (titles = Anthropometric Data and Vital Signs: All Subjects,
         varlist = Age Hgtno Wgtno Tmpno RsprRt HrtRtno SystBPno DiasBPno)

```