Dynamic Screen Generation Using Formats:
The Case Against Hard-Coding
Louise Weiler, Pacific Gas and Electric Company

ABSTRACT
All of us who have ever had to modify a SAS® program know that making one little change "here" can affect something we haven't anticipated "there." SAS® systems are particularly vulnerable to change because of the complex relationship between screen variables, macro variables, SAS variables, and, in Version 6, SCL variables. Making a little change "here" to a screen can have a devastating effect "there" in the behavior of the system.

Within ordinary SAS programs, you can sometimes avoid having to make code changes by using table lookup techniques. These techniques are widely recognized to improve both efficiency and maintainability. Within SAS/AF systems, the same principle applies—you can avoid making those little changes to screens by dynamically generating the screens rather than hard-coding them.

INTRODUCTION
The purpose of SAS/AF systems is to provide users with choices that can be made on a screen. The choices can be made by checking off a list of options, filling in blanks, and so on. Usually, when a list of options is presented to the user, the list is static; once the system has been designed and built, this list cannot change unless the programmer changes the system. In some SAS/AF systems, however, the nature of the options is such that some will become obsolete and new ones will need to be added. If you can anticipate these changes, you can build automatic ways of incorporating them into your system.

This paper will compare dynamic screen generation versus hard-coding as a means of incorporating change in a SAS/AF system. The paper begins with a discussion of which type of SAS/AF screens are good candidates for dynamic screen generation. Then, two examples will be developed, one demonstrating hard-coding, the other demonstrating dynamic screen generation. Each example will show how to make one change to the screen. Finally, the paper will reveal the true power of dynamic screen generation—you can build a system that is entirely table-driven.

The issues addressed in this paper apply to any SAS/AF system, regardless of whether you use Version 5 or Version 6. The examples, however, will be presented using Version 6. Dynamic screen generation can be accomplished with much greater ease and speed in Version 6 than in Version 5. This paper assumes a basic understanding of building a SAS/AF PROGRAM entry. The reader is referred to the SAS/AF® Guide for Personal Computers for review. Note: All the examples in this paper have been pared down to make them simple to understand. They do not represent a full system.

RECOGNIZING APPROPRIATE SCREENS
The decision to use dynamic screen generation rather than hard-coding depends in part on the likelihood that a screen will need to change over the life of the system. The following three screens each contain a list of options that the user can check off. Each list of options has, however, a different likelihood of needing change, ranging from very little to very large.

Screen 1 of the Tourist Advice System displays the list of fundamental tasks that the system will perform. There is little likelihood that this screen will need to change over the life of the system.

Screen 2 is more likely to need change than Screen 1 if there is a possibility of expanding the system to include, for example, a "Sightseeing" or "Shopping" option. This screen will be reviewed again at the end of this paper.

As the entertainment "scene" in the city inevitably changes over time, the items on Screen 3 must also change if the Nightclubs data base is conscientiously...
maintained. The following two examples will show the steps necessary to add "Comedy" as an option to the list of types of nightclubs on Screen 3.

HARD-CODING

Currently, there are only three items on Screen 3. If we had originally been tempted to hard-code those items on an entry called HARDCODE.PROGRAM, the display panel would now look like Figure 1.

Figure 1

It is a simple task to insert a line between "& Blues" and "& Jazz" to add "& Comedy"; however, your work is not finished. You will also have to modify the "main" section in the SCL source panel (see Figure 2).

Figure 2

The "main:" section of the SCL source code performs the following:
1. Determines which field on the screen has been marked.
2. Associates that field with an SCL variable, "type:"
3. Passes the value of "type" to "&type" in SAS code.
4. Submits the SAS code.

The SAS code selects and prints the appropriate nightclubs based on the statement 'WHERE TYPE = "&type":' Note that the SAS variable TYPE contains a coded value for type of nightclub which is associated with a format $NITEFMT in the data set SUGI.NITECLUB.

In order to add "Comedy", the programmer will have to insert a line of code in the "main:" section, change the numbers on the "field" variables, and put in the correct code 'CO' for "Comedy" that occurs in the data base.

DYNAMIC SCREEN GENERATION

Dynamic screen generation is based on the following premise:

Fields in SAS/AF screens can be filled in with values that come from a SAS data set.

The extended tables feature in Version 6 accomplishes this nearly automatically.

Building the Data Set

Before you can perform dynamic screen generation, you need to build a data set containing the values to fill in the fields on the screen. In order to generate a screen that would look like Screen 3, the minimum you would need in a data set (see Figure 3) is one variable and three observations.
Using the extended tables feature, you could move the values in the variable CLUBTYPE in this data set into a field called &CLUBTYPE on the screen. In order to add “Comedy” to the screen, you would need only to add a “Comedy” observation to SUGI.CLUBTYPE. While the modified data set would be sufficient to generate a modified screen, you would still have to modify the SCL source code. The SCL source code would still be responsible for associating responses on the screen to a value of the variable TYPE through a series of “if...then” statements. What’s needed is a technique to make the SCL source code table-driven at the same time that the screen is being dynamically generated.

The key to doing this is to build a data set that also associates the responses on the screen to a value of the variable TYPE, so that the value can be passed in a macro variable to the SAS code in the submit block.

In Version 6, this is accomplished using PROC FORMAT with a CNTLOUT option (see Figure 4). The CNTLOUT option specifies an output data set that contains information about the format.

**Figure 4**

```
* Code to create format $NITEFMT, *;
AND data set SUGI.NITEFMT *;
***************************************************************************************;
PROC FORMAT LIBRARY=LIBRARY;
  VALUE $NITEFMT 'BL' = 'Blues'
                  'JA' = 'Jazz'
                  'RR' = 'Rock' n' Roll'
RUN;
PROC FORMAT CNTLOUT=SUGI.NITEFMT LIBRARY=LIBRARY;
SELECT $NITEFMT;
RUN;
```

The output data set (see Figure 5) contains one observation for each range defined in the format. The variables of interest are START and END which are the range starting and ending values, and LABEL which is the formatted value. Note that the values in START and END are equal. The variable FMTNAME contains the format name and the variable TYPE indicates that this is a character format. (Don’t confuse this variable TYPE with the variable TYPE in SUGI.NITECLUB.)

**Figure 5**

```
+-------------------------+
<p>| Proc Print             |
| Selected Variables     |</p>
<table>
<thead>
<tr>
<th>Data Set SUGI.NITEFMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSS</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>
+-------------------------+
```

The data set SUGI.NITEFMT in Figure 5 can be updated by modifying and running the PROC FORMAT in Figure 4. PROC FORMAT also allows the reverse approach—you can first create or modify the data set described in Figure 5 and then use the data set to generate the format. This is accomplished with the statements:

```
*** Code to create a format from a data set;***
******************************************************************************;
PROC FORMAT CNTLIN=SUGI.NITEFMT LIBRARY=LIBRARY;
RUN;
```

The data set SUGI.NITEFMT will be used for the following example of dynamic screen generation.

**Building the Entry**

The display panel for entry DYNAMIC.PROGRAM will now be coded once, and only once, regardless of how many nightclub categories need to be displayed. The entry will employ the extended table feature. To designate a display panel for extended table processing, you must mark the Extended Table attribute in the GATTR panel of the entry.

**Figure 6**

The extended table feature allows you to repeat a field or set of fields defined in the display panel. The portion of the panel below the ^ and defines the fields that will be repeated (see Figure 6). This is a scrollable section and is referred to as the extended table. The portion of the display panel above the ^ is nonscrollable.

The fields in the extended table will be filled in with the values in data set SUGI.NITEFMT. The field "&label" will be filled in by the variable LABEL. The "&start" field will be filled in by the variable START. The "&start" field should be designated as a non-display field in the ATTR panel so that you will have access to it in the SCL code but the user will never see it.

**Non-display: X**

The SCL source code (see Figure 7) controls the processing of the extended table in the "init:" and "getrow:" sections. The code in the "main:" section still handles passing the value of the SAS variable TYPE to the
SAS code, but this is accomplished without any "if...then" statements. Instead, the SCL code searches for the row in the extended table that the user has selected. The value of &start in that row is passed to the SAS variable “TYPE.”

Figure 7
***********************************************;
1< SCL source code for screen DYNAMIC. PROGRAM *;
***********************************************,
]init:
 dsid = open(~sugi.nitefmt'};
 nobs = nobs(dsid);
 call set(dsid);
 do i = 1 to nobs;
 rc=fetchobs(dsid,i);
 end;
 return;

main:
do i = 1 to nobs;
 if action ne _blank_ then
 submit terminate;
 PROC PRINT DATA=SUGI.NITECLUB NOOES;
 WHERE TYPE"&start";
 FORMAT TYPE $NITEFMT.;
 TITLE 'Nightclubs in San Francisco';
 RUN;
 endsubmit;
end;
 return;
term:
 return;
get row:
 ro = fetchobs(dsid, Currow);
 return;

Changing the Entry
In order to add "Comedy" to the screen DYNAMIC. PROGRAM you must perform the following steps:

1. Modify the format $NITEFMT.
2. Regenerate SUGI.NITEFMT.

You do not need to modify the display panel or the source panel to incorporate this change.

HARNESSING THE POWER OF DYNAMIC SCREEN GENERATION: A BRIEF LOOK

Recall for a moment the Category Menu (Screen 2) in the Tourist Advice System, on which the user may choose either "Restaurants," "Hotels," or "Nightclubs." If the subsequent "Restaurants" and "Hotels" menus look similar to the "Nightclubs" menu, you should consider designing one generic screen entry for all three rather than separate screens for each. You will simplify your system by having two fewer screens.

You can convert DYNAMIC. PROGRAM to GENERIC. PROGRAM that can be used by restaurants and hotels as well as nightclubs. In order to do this, you need to set up certain conditions. One approach is to do the following:

1. Put the tourist information for restaurants, hotels and nightclubs in separate data sets, i.e., SUGI.RESTAUR, SUGI.HOTELS and SUGI.NITECLUB.
2. Assign the name "TYPE" to the variable that describes the type of establishment in each of the above data sets. For example, in the restaurant data set, the variable "TYPE" would contain coded values for "Chinese," "American," "French" and so on.
3. Provide a separate format for the variable "TYPE" in each data set, e.g., $RESTFMT would be the format for type in SUGI.RESTAUR.
4. Generate a separate SAS data set for each format using the CNTLOUT option of PROC FORMAT, e.g., generate SUGI.RESTFMT from $RESTFMT using code similar to Figure 4.

Assume for now that when a user selects either restaurants, hotels or nightclubs on the Category Menu, all appropriate data set names and formats are placed in macro variables. GENERIC.PROGRAM can then retrieve the data set names and formats from the macro variables and use this information to place on the screen the appropriate title and list of items to be selected (see Figures 8 and 9). If the user selects "Restaurants" on the Category Menu, then the GENERIC.PROGRAM screen will appear to the user as "Restaurants Menu", listing the different types of restaurants.
When you choose the latter, you have harnessed the ultimate power of dynamic screen generation—you have created a table-driven system. In a table-driven system, the flow of control is contained in tables, not in code. A detailed treatment of table-driven systems would be a good topic for a SUGI paper in the future.

**CONCLUSION**

You do not have to be locked into the original options you envisioned for your SAS/AF system. Dynamic screen generation makes SAS/AF systems easy to change and to expand. You will appreciate the ease and the power of this technique, and your users will appreciate the results.

If you would like an example of this technique using Version 5, or if you have any comments, feel free to contact the author at:

Louise Weiler  
PG&E  
77 Beale St. Room 1006  
San Francisco, CA 94106  
(415) 973-5485

**Acknowledgements:**

Thanks to Peter Kretzman, Don BeLow, Stan Sibley, Judy Alicante, Steve Marino and Danine Cozzens for their invaluable contributions.

**Note:** SAS and SAS/AF are registered trademarks of SAS Institute Inc., Cary, NC, USA.