Writing SAS/AF Applications that run in both an Interactive and Batch Environment

Jim Nelson, MOS-11, Motorola, Austin, TX
Matthew Grover, J JT Consulting, Austin, TX

ABSTRACT

Many SAS/AF applications have been written that run in an interactive mode but cannot run in a batch mode. Other applications have been written to run in both interactive and batch modes; however, most often these applications generate static code, using only simple code substitution. A major SAS/AF application has been written that runs in both interactive and batch modes. This paper presents a design for creating SAS/AF applications that run in both interactive and batch modes. The application is a dynamic code generating system, generating and executing dynamic SAS code in both modes.

INTRODUCTION

Most systems which get developed at Motorola's MOS-I I wafer fabrication facility are first conceived to run in an interactive full screen mode. However, even before the system is put into place, there comes a ground swell of requests that the system run in a batch mode as well as an interactive mode. Simple applications can meet this requirement. It is when the application becomes complex and requires conditional code generation that a problem arises. There is an inequality in the capabilities between the two modes. SAS provides a very good system for conditional generation of code in an interactive mode via SAS/AF's Screen Control Language, but only a limited batch code generator using the Macro Language. Any SAS/AF system that uses program entries or frame entries cannot run in batch mode, and the macro language is severely limited in its functionality. We have overcome these problems in the past by developing different sets of code to allow the system to run in both modes, and/or using macro code that is called by the SAS/AF frames in interactive mode or by statements in batch mode. Both of these approaches have their limitations. Double coding is not an efficient way to develop and maintain code. Confining a system to only being able to use macros to generate needed code does not take advantage of programming features in SCL.

We were given the challenge to solve this issue with one of our large SAS based systems called EDAS. This system is similar in design to SAS/ASSIST in that the user makes selections from program screens, choosing the data set, options, and parameters necessary for the particular program. The system dynamically generates the SAS code and submits the code for processing. Batch mode processing of the EDAS code has been limited to having the system save code generated to an ASCII file and using static SAS code in batch mode. The only item that was interchangeable from execution to execution was the name of the data set. This was accomplished by using a macro variable as the data set name, and either manually editing code and changing the variable value, or by using a separate interactive program to set the value of the macro variable. Our users had a need to dynamically generate scripts in batch mode.

Our first task was to find out from the users what part of EDAS needed to be available in dynamic batch mode. The user community's response was 'All of it!'. Rewriting all of EDAS into a dynamic, batch code generating program, along with maintaining the interactive version was not practical. A versatile solution had to be found.

The solution had to allow for the use of the same code in both batch and interactive modes. The solution needed to be able to dynamically generate code and submit it for processing. To meet the system requirements required the use of functions found only in SCL.

The solution involved code isolation and the use of lists for intermodule communication. The solution design is composed of three components: 1.)
frames/program entries 2.) global lists 3.) methods. Basically, all of the code generation was removed from the program and frame entries and moved to SCL methods. The purpose of the program and frame entries is to gather and validate users input. The passing of all the varieties of user input is accomplished through a structure of global lists. The global lists hold printer parameters, data set names, and other sublists that possess information necessary for writing SAS code. The SCL methods isolate all code generation and submit code to SCL entries. The SCL methods can be run in both and interactive and batch mode. In batch mode, the same user input can be input from a configuration file and converted to an SLIST entry by a master SCL program that executes a PROC DISPLAY. This program calls the same methods as are used in the interactive version of the system.

FIGURE1: Application Design

I. FRAMES
The major function of the frames is to be used as an interface for populating and editing the global lists. The information the user puts into the list will actually be used to write SAS/BASE, SAS/STAT, or SAS/QC code. There are several benefits to using this type of design for list maintenance. First, you have the ability to dynamically write your own SAS code without having to know anything about SAS. All that is required is that you know what type of procedure you want to run. Second, user input can be validated before code is actually submitted for processing. Third, the list can be saved as an SLIST entry and retrieved in a subsequent SAS session.

Every frame entry is comprised of multiple widgets that fall into three categories: manipulation widgets, unique widgets, and 'shared_variable' list widgets.

Manipulation widgets may include push buttons, control objects, scroll bars, or container boxes. The purpose of manipulation widgets is to provide you the ability to manipulate how information is displayed to you on the screen. The values returned from these widgets have no affect upon submit code. Therefore, the names and values of these widgets are not saved in a list.

Unique widgets are unique to each screen and are most often of the type efield, radiobox, or checkbox. Unique widgets are classified as a separate group because the values returned from these widgets are valid SAS statements that are saved in the 'widgets' sublist. The 'widgets' sublist is updated every time you change the value of one of the widgets either by clicking on the widget or typing something, if the widget is a text entry field.

Shared_variable list widgets on each frame are used to display the names of the variables in the 12 variable list that are associated with that frame. The shared_variable sublists are sublists of variables that are shared across all frames in the application. Because of the design of this application, in which you are allowed to perform a variety of different analyses during one SAS session, it is imperative that the variables you select for one frame be saved and transferred to the next frame. This greatly reduces
the amount of time you spend selecting variables. A frame that does 'Simple statistics' may utilize only two of the twelve variable lists while a frame that does 'Analysis of Variance' may use six of the variable lists. The shared_variable sublists can be populated and updated in two different manners. The first manner allows you to click on a push-button to display a listbox containing the names of variables that can be selected. You also have the option of hand entering the variable names. If a text entry field is modified, a method is called to check to see if the variable name in that text entry field exists in the data set and that the variable is of the appropriate type. If the variable exists and it is of the correct type, it is then added to the associated variable list. However, if the variable does not exist in the data set or is of the wrong type then a listbox containing the names of variables for that data set is displayed.

II. LIST STRUCTURE

The global list is comprised of two sublists: one which contains shared variable lists and another list which contains sublists that are unique to the frame.

A 'Shared_Variables' list was created to hold the separate variable sublists. Given that a different analysis may require more variables and may also require variables of different types, the variables are grouped into logical lists. In the current application 12 variable lists are used: 'varlst1-varist4' are lists of response variables; 'bylst1-bylst4' are lists of by-group variables; 'altlst1-altlst4' are lists of variables that do not fall into a 'varlst' or a 'bylst'. Each of 12 variable lists holds two sublists ('vars', 'desc'). The 'vars' list contains a list of variables that were selected for that list. The 'desc' sublist contains items that describe the information in the 'vars' sublist, such as of the type of variables that a user can put into the:

Sample Shared Variables List:

shared_variables=(varlst1=(vars=(P3004=1 P3003=2 P3006=3 P3005=4) [4]) desc=('screen' 'N') [32]) [2]

The 'widgets' sublist contains two unique lists. These sublists are cleared every time a frame is exited. The first sublist named 'name_value' actually contains sublists that are named by programmer. These sublists contain named items which are the names of the widgets. The value of the items in the lists are the return values of the widgets. These lists allow the programmer to group widgets into any named sublist of their choice. The 'name_value' lists have no affect on the frame entry, they are merely ways of grouping logical groups of widgets, so that processing of widget values later on is simplified. The second sublist under the 'widgets' list is the 'type_of' list which has named items which are the names of the widgets and the values of these items are numeric or character. If the widget type is radiobox or checkbox the item value is the value of current station for that widget. If the widget type is text entry then the value of the item is the value of text entry field.

Sample Widget List:

widgets=(name_value=(stats=(mean = 'mean' std = 'std' n = 'n')) [200]) [190] type_of=( mean=1 std=1 n=1 ) [188] [180]
III. Methods

The methods consist of submit code that declares the procedure(s) to be run. The method can be passed one optional parameter 'slistname'. The 'slistname' parameter is a four-part slist entry name that was created while the user was running the application in an interactive mode. If the optional parameter 'slistname' is not passed, then the list already exists in memory. In batch mode the method is passed the name of an SLIST entry. The method in turn calls another method that executes a 'fillist' function before the submit code is placed in the preview buffer. When the method is executed in batch mode via an SCL entry that runs a PROC DISPLAY.

A.) Interactive Mode Sample Execution:

1.) Frame:

The following frame allows you to run a MEANS procedure for selected response variables and group the selected summary statistics by specified output variables

FIGURE2: Sample Frame

2.) The Global Lists

After you have made the selections that are displayed on the frame, the shared_variables and widgets sublists have been populated with given values.

shared_variables=(varlst1=(vars=(P3004=1
P3003=2
P3006=3
P3005=4
) [4]
desc=('screen'
'N'
) [32]
(bylst1=(vars=(wafer=1
) [60]
desc=('screen'
'C'
) [62]
) [2]
widgets=(name_value=(stats=(mean = 'mean'
std = 'std'
n = 'n'
nmiss = '
sum = '
min = '
max = '
range = '
cv ='
) [200]
type_of=(
mean=1
std=1
n=1
nmiss=0
sum=0
min=0
max=0
range=0
cv=0
) [180]

3.) The Method

When you select 'OK' from the pull down menu the SCL compiled with the frame calls method 'STATS'. Under interactive mode the 'slistname' parameter is not passed to the method.
STATS:
method optional=slstname $ 35;
call method ('init.scl', 'procinit')
    submit;
    proc means
    endsubmit;
call method ('utils.scl', 'subwl-
list', 'test');
endmethod;

4.) Code Generated
The following code was generated when method STATS was called.
data test;
    set die.mydata;
    keep wafer p3004 p3003 p3005 p3006;
    proc means n mean std;
    var p3004 p3003 p3005 p3006;
    by wafer;
run;

B.) Batch Mode Sample Execution

1.) Master SCL program
The master SCL program is a SCL program that serves as a dynamic code interpreter, that reads and executes scripts. The scripts contain multiple lines of code that are all preceded with a '#' symbol. The word following the '#' symbol is an application defined function. The master SCL program parses the code looking for the '#' symbols. The dynamic code interpreter calls the functions which in turn perform some variety of SAS code generation or execution.

• #LIST function populates a named variable list with named item.
• #WLIST function populates a named widget list.
• #RUN function executes an a SCL method, passing any parameters needed.

2.) The SLIST entry and Method
When the #RUN function is executed it calls method 'STATS' and passes the name of a SLIST entry. This SLIST entry is then copied into the global environment list. The list structure has the same format as the interactive mode list structure.

3.) Code Generated
The code generated is identical to the code that was generated in interactive mode.
The master SCL program is a very powerful mechanism for executing a multitude of procedures with very minor code modification. The following code uses the same variables across several methods. If the variables or widgets need to change for a method, all that needs to be done is to add another '#LIST' or '#WLIST' function call to the code. An example of how this code would look follows:

```sas
#LIST varlst1 p3004 p3003 p3005 p3006
#LIST bylst1 wafer
#WLIST stats mean n std
#RUN
('procs.scl', 'freq', edaslib.die.xlist.slist')
```

1*) Run a correlation
```sas
#LIST varlst1 p3004 p3003 p3005 p3006
#LIST bylst1 wafer
#WLIST corrtype pearson
#RUN
('procs.scl', 'corr', edaslib.die.xlist.slist')
```

1*) Run a univariate
```sas
#WLIST unv_plot normal plot
#RUN
('procs.scl', 'univar', edaslib.die.xlist.slist')
```

1*) Run a frequency
```sas
#RUN
('procs.scl', 'freq', edaslib.die.xlist.slist')
```
Conclusion

The components of the application described in this paper have many useful features. The lists allows the programmer the ability to easily validate user input before the code is submitted. The lists that are created during an interactive mode can be saved and retrieved during a subsequent batch or interactive SAS session. The frames provide the user with the ability to populate lists with values returned by the widgets they have selected. The values returned from the widgets are actually SAS code; in essence you are dynamically writing SAS code without having to know any SAS code. The methods are modular enough so that multiple applications could use the same method to perform similar tasks while simultaneously providing different interfaces to the user. Most of the SAS code that is submitted comes from lists, thus minimizing the amount of actual hard code in the methods.

Code duplication is eliminated for batch and interactive modes with the exception of one master program used in batch processing. The scripts are generated dynamically. Since the code generated for one variable can be drastically different then the code generated for multiple variables, the use of dynamic scripts allows you to maintain only one batch script that can be modified very simply to generate far different code. Finally, the complexities of code generation are not passed on to the user.

Having SAS/AF applications that run in both an interactive and a batch environment reduces duplicate code, saves time by allowing users to queue up multiple programs and run them in batch, and decreases the amount of code maintenance required.

SAS, SAS/AF, SAS/ASSIST, SAS/QC, SAS/STAT are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries.