Tips From A 1st Time FRAME Application Developer

Robin Reese, Chevron Information Technology Company, San Ramon, CA

Abstract

The following paper describes tips and techniques that proved helpful in developing a FRAME application under Windows®.

Introduction

Building an application from scratch can be a programmer's dream or nightmare. Being unfamiliar with the language can add an interesting twist. Add the fact that the FRAME entry is fairly new, and the words "opportunity" or "challenge" readily come to mind.

We have built an application using SAS/AF® with FRAME entries , and a host of other SAS System products. Even though we were unfamiliar with SAS/AF or even the SAS language, we found that the SAS System offered several ways to make programming easier as well as ensuring a solid application.

Following are some tips that we came up with in developing our first FRAME application using SAS/AF in Windows.

Program Development

1. Document.

Sorry, it needs to be said. No matter how familiar you are to a language and application, the next person to support the application may not have your expertise.

Also it is very helpful to document where things are. In which directories are your programs, data sets, resources stored? Where is your development version? Production? Document this information before you start to develop.

2. Use the colored text option to highlight changes.

When modifying existing SCL code, it can be helpful to modify the color of the new code to highlight the changes. This allows the eye to easily spot code changes. For example if the original code is in yellow, the modifications can be made in green.

To change the color of the text being typed with pmenus:

View ==>Colors==> Text==> Green

To change the color using the command line :

COLOR TEXT 'G' where 'G' is the color green.

When the modifications are tested and ready for production, the code modifications can changed to the original color by highlighting the text with the mouse and issuing the following command from the command box or line:

MTEXT COLOR = 'Y'

This will changed the marked text to the color yellow, which in this case was the original color.

The SAS System is very liberal about setting up colors for backgrounds, text, messages etc. Use these options to customize a desktop for your benefit. However, if there are multiple developers, be respectful of pieces that will be shared (i.e. SCL code). What may look like cheery, colorful code, may give your teammates migraines. Also beware of setting the background color and the text color the same for obvious reasons.

Colors available are:

Color	Code	Color	Code
Blue	'B'	Red	'R'
Green	'G'	Cyan	'C"
Pink	'P'	Yellow	'Y'
White	'W'	Black	'K'
Magenta	'M'	Gray	'A'
Gray	'A'	Brown	'N'
Brown	'N'	Orange	'O'

3. Make ample use of control labels.

Control labels allow code to be called from several different places within the same program thus reducing redundant code.

In the following example, the CLOSEOUT control label is linked (or called) to close data sets before paging forward or backward.

```
/*********************CLOSEOUT*******************/

CLOSEOUT:
                              /* close list data sets*/
if (dsid   > 0)   then rc=close(dsid);
if (facdsn > 0)   then rc=close(facdsn);
if (stodsn > 0)   then rc=close(stodsn);
if (directry > 0) then rc=close(directry);
return;

/*********************PAGEBACK*******************/

PAGEBACK:
                              /* close data sets */
link CLOSEOUT;
                              /* save the selfac and stock  to
                                 be passed to the next screen
                                 */
call symput('selfac',selfac);
call symput('selstock',selstock);
                              /* call the next screen */
if pageno = 'Page: 2' then call goto('season1.frame','r');
return; /* pageback */

/*********************PAGEFOR*********************/

PAGEFOR:
                             . /* close data sets */
link CLOSEOUT;
                              /* save the selfac and stock  to be
                               passed to the next screen */
call symput('selfac',selfac);
call symput('selstock',selstock);
                              /* go to next screen */
if pageno = 'Page: 1' then call goto('season2.frame','r');
return; /* pagefor */
```

4. Make ample use of method blocks.

A method block is a set of SCL code that is stored in an SCL entry. Methods allow code to be called from several different SCL programs. This can eliminate redundant code when similar functions are performed in multiple FRAME entries. Variables can be passed in and out of methods blocks, allowing the method block to use, modify, and/or return a value to the calling entry. Methods are easy to understand and use and support modularity concepts.

Calling routine: (MAIN.SCL)

```
INIT:
return;

                                  /*Executed on exit*/
 TERM:
return;

 LOADFORE:
                                  /* Cleanup  big files that hog
                                 hard disk space */

call method('utility.scl','cleanup');
return;
```

Called Method (UTILITY.SCL)

```
                                 /* another example of a
                                    method */
ADD:
 method;
 put 'another method';
 endmethod;
                                 /* this method cleanups up big
                                    temporary files that hog
                                    hard disk space */
                                  /* Called by: Main.scl
                                     Relation.scl */
CLEANUP:
 method;
 rc = delete('work.allkids');
 rc = delete('work.total');
                                 /* cleanup old statistics from
                                   old forecasts */
submit continue;
data demand.mape(alter=XXXX);
set demand.mape;
 delete;
 run;
 endsubmit;
 endmethod;
```

5. Stay away from %INCLUDES in SCL code.

The %INCLUDES statement allows code to be stored in external file and executed when the %INCLUDE statement is hit. When using the SCL debugger, the line stepping does not access the %INCLUDEd code and the line numbers get out of order. Method blocks code will be pulled into the debugger and are execute correctly.

The method blocks perform the same functions as the %INCLUDE and are cleaner and easier to follow.

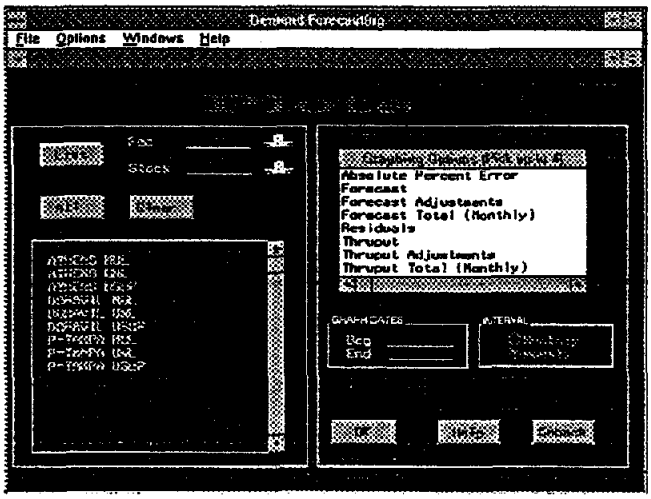
6. Design reusable FRAME entries.

In our application we have two screen designs that account for 90% of the FRAME entries. See Display 1. The left side of our screens perform a similar function of selecting cities and petroleum products. The right side is used to perform the

action on what was selected on the left-hand side. In this example, the action to be performed is plotting.

By creating template screens, the FRAME and the SCL code are copied when a new screen is needed. The basic functionality of the screen is already in place. The only modifications necessary are the functions that are unique to the screen.

The users benefit by having screens that are similar in design. The layout including buttons, text entry fields, etc. are always in the same place. There is nothing more annoying than having to search for an "OK" or "Cancel" button on a screen.



Display 1

7. Set FRAME design standards.

A good reference for GUI standards is "The Microsoft Guide to Designing the User Interface" by Microsoft Press. Make the colors meaningful, i.e. blue for enterable fields, yellow for unenterable. Some colors project better than others, stay away from colors that are hard to see. Also take into consideration that many people are color blind.

8. Use Region Attachments.

Region attachments automatically resizes or moves a region when it is attached to another region. It may not seem useful at first, but spend a couple of hours aligning push buttons and container boxes and the use becomes glaringly obvious. An excellent paper was presented at SUGI 19 by Bud Whitmeyer highlighting region attachments.

Miscellaneous

9. Iconize or close the Log, Program Editor and Output in AUTOEXEC.SAS or CONFIG.SAS

The SAS System and Windows takes time to redraw the log, program editor and output. By closing or iconizing them, the application FRAMEs display faster. Our clients noticed a significant improvement when we added this feature. The application is also cleaner when the log, program editor and output screens are not flashing between FRAME entries.

Example:

```
/*This file sets up SAS for Demand Forecasting, and then
starts*/
/*Demand Forecasting.*/

libname analysis V608  'c:\sas\analysis';
libname demand  V608  'c:\sas\demand';
libname forecast V608  'c:\sas\forecast';
options awsdef = 0 0 100 100;
                    /*  iconize the program editor and log */
dm 'pgm; icon';
dm 'log; autoscroll 0; icon';
dm 'output; zoom on; color background W; color byline K;
color message R; autoscroll max;';
dm 'af c=forecast.appl.startup.scl aws=no' continue;
```

10. Be prepared when calling the Institute for help.

When calling technical support, set up a simple example of the problem you are trying to solve. Use objects in the examples that are easier to understand and will not deter from the problem. For example, use cities and fruit instead of 10/30 oil or hydra weight-150.

By taking the time to set up the example, the solution to the problem usually becomes apparent.

11. Password protect data sets.

Password protecting the application data sets can save time and effort if you or your clients are interested in working with the application data sets outside of the application. Too many times critical files can be destroyed by a small syntax error. In the worst case, it could take hours or days to recreate a dataset that has not been backed up properly.

To password protect a dataset:

```
DATA  CRITICAL.CRITICAL(WRITE=WHITE
ALTER=AMBER);
SET CRITICAL.OLD;
RUN;
```

To use a password protected dataset:

```
DATA CRITICAL.CRITICAL(WRITE=WHITE
ALTER=AMBER);
IF  USELESS = 'Y' THAN DELETE;
RUN;
```

The write password protects against changing the data in the file. The alter password protect against deleting, replacing or modifying variable attributes. There is also a read password

that protects against reads. (Please see SAS Technical Report P-222, page 11 for more information).

12. Use the SAS/FSP procedure FSLIST to display external help files.

Our help files are in ASCII files maintained by WordPerfect® for Windows. Instead of having a set of on-line help and external help, that would have to be maintained. We use one set of text files for both. This ensures that both the on-line and external documentation are always in sync.

Example:

```
HELP:
proc fslist fileref='C:\SAS\FORECAST\HELP\EXPORT.DOC';
 run;
return; /*HELP*/
```

13. Customize PMENU entries for your client's needs.

Use the PMENU facility to allow access among the FRAME entries. You can also use the PMENU function to allow the client access to pieces of the SAS System outside of your applications. On the other hand, access can be restricted by not providing these functions.

Conclusion

Most of what we learned in developing our application is from common sense, but a lot of it came from our noses firmly entrenched in the SAS System manuals. Always keep your eyes open for ways to make programming easier. The best ideas come when you aren't looking for them.

References

SAS Institute Inc., SAS/AF Software: FRAME Entry, Usage and Reference, Version 6, Cary NC: SAS Institute, March 1993

SAS Institute Inc., SAS Language Reference, Version 6, Cary NC: SAS Institute., June 1992

SAS Institute Inc., SAS Screen Control Language, Version 6, Cary NC: SAS Institute, November 1991

SAS Institute Inc., SAS Technical Report P-222, Changes and Enhancements to Base SAS Software, Release 6.07, Cary NC: SAS Institute, October 1991

SAS Institute Inc., SAS Technical Report P-216, SAS/AF Software, SAS/FSP Software, and SAS Screen Control Language: Changes and Enhancements, Release 6.07, Cary NC: SAS Institute, October 1991

Acknowledgment

I would like to thank Daphne McCarley for many of the ideas presented and also her unrelenting scrutiny of this paper.

SAS, SAS/AF, SAS/FSP are registered trademarks or trademarks of SAS Institute Inc., in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

Contact Information

Robin Reese
CP K2280
6001 Bollinger Canyon Rd.
San Ramon, CA 94583

(510)842-4181
TRLMU@CHEVRON.COM