Writing Efficient Screen Control Language Programs
Annette Harris, SAS Institute Inc., Cary, N.C.

Screen Control Language was introduced in Version 5 of SASIAF® and SAS/FS® software. It provides new power and flexibility to full-screen applications. Version 5 SASIAF applications are converted to Version 6 format using the CPORT and CIMPORT procedures or the VSTOVC procedure. But there are some techniques that you can use to ensure that your application is most efficient, whether it is a new Version 6 application, or one converted from Version 5.

In Version 5 of SASIAF software, the DISPLAY procedure was used to move from screen to screen in a full-screen application. The code generated by the Version 6 compiler produces PROC DISPLAY statements. To take advantage of the new performance efficiencies, it is recommended that where possible, you delete PROC DISPLAY statements and use the AF command to invoke your application. The SAS® System must be invoked with the DMS option in order to use the AF command.

```
AF catalog-name.catalog-name.stats;
```

or

```
CD 'AF catalog-name.catalog-name.stats' AF;
```

The CALL DISPLAY routine would then be used to move between windows. This means the procedure does not have to end, submit SAS statements, and then be reinvoked in order to continue processing. Obviously response time between windows should be faster. Note that the PROC DISPLAY is still available if you execute the SAS System in a text-mode environment.

When converting, you must decide whether you will be using PROC DISPLAY or customizing your application to take advantage of the AF command and CALL DISPLAY throughout your application. Because PROC DISPLAY and the AF command use separate windows, using these commands inconsistently will affect when SAS statements are submitted and possibly the flow of your application.

Depending on the actual statements in your source program, you may also find that there are SCL functions that can reduce the amount of coding required to perform a task. For example, in Version 5, if you had two user fields that were mutually exclusive and responded with a customized message if both fields had values, you could use the FIELD macro. In Display 1 below, FIELD1 and FIELD2 are the associated macro variable names for user fields.

```
%SYSDisplay
    FIELD1 = "FIELD1"
    FIELD2 = "FIELD2"
    IF FIELD1 = "FIELD1" AND FIELD2 = "FIELD2"
        &SYSMSG = "FIELD1 and FIELD2 are mutually exclusive"
    END;
```

Display 1 Example Version 5 Code for Validation

In Version 6 SCL, the same validation can be done and macro variables are not required, as shown in Display 2 below.

```
%SYSDisplay
    IF FIELD1 = "FIELD1" AND FIELD2 = "FIELD2"
        &SYSMSG = "FIELD1 and FIELD2 are mutually exclusive"
    END;
```

Display 2 Example Version 6 SCL for Validation

(Note: the special system variable _BLANK_ is available only in SASIAF software.)

This brings us to features that you can use that will assist in making your application efficient and useful to your users: techniques that increase productivity while developing the application, designs that ensure the best execution performance, and enhancements that provide support to your users—saving you time in the long run.

INCREASING DEVELOPMENT PRODUCTIVITY

You can take several steps while developing an application to make the best use of your development time. Some of these include:

* Use the SYSMSG function to provide information about the return codes from various SCL functions. SYSMSG will provide the error message produced by the SAS System. Before creating the production version of your application, you will want to remove many of the SYSMSG inquiries because you will not want your users to see messages they might not understand.

* If you choose to use the information produced by the return code, it is better to continue to use the same variable repeatedly. While the development version of your application may use several variables to monitor the return code of various SCL functions, consider modifying the source to check the same variable when creating the production version.

* You may also find that it is easier to use the NOMSG option on the PROC BUILD statement so that all notes and warnings generated by the compiler will be written to the SAS log rather than to the MESSAGE window. When you end the BUILD procedure, NOMSG will ensure that compile information is still available in the SAS log.

* If you use the LOG command to check the log for messages, the NEXT BUILD command can be used to return to your PROC BUILD session. Note that there is a
difference between using the NEXT command and the
SWAP command. While NEXT may move you between DM Windows, SWAP will jump you between AF windows only.

- When you provide default or null character arguments to functions, the appropriate argument is `"", not "."

- If you are planning to use the SCL Source Level Debugger during a PROC FSEDIT session, use the PUT ALL statement in the FGSCINT reserved label. This will initialize all variables for the debugger. Remove the PUT statement when you have completed debugging.

- For key-stroke efficiency, it is better to use the same catalog to contain all entries used in your application. Then you have only to specify a 2-level (entryname.entrytype) instead of a 4-level (libref.catalog.entryname.entrytype) name.

- To decrease the amount of disk space your production version requires, use the NOSOURCE option on the MERGE statement in PROC BUDL when copying your application to the production library. It is important to save an original catalog or the SCL in a file because, as NOSOURCE implies, the SCL source is not available for editing when a catalog is created and this option has been specified.

ENHANCING PERFORMANCE

The hundreds of SCL functions available make it possible to accomplish many tasks with a limited number of statements. To maximize use of these functions, consider the following:

- When you open a data set, check to see that it opened successfully. In many cases, you will not want to continue processing in the same manner if the data set failed to open.

- When providing arguments to SCL functions, be aware of the differences that exist in quoting the arguments. In most cases, this is the difference between passing a variable name and passing a variable value. Suppose that you want to allow your users to specify the data set they want to open, and the name of the window variable is DATASET. You would code the OPEN function as follows:

  ```sas
  disp = open(dataset,"r");
  ```

  If you used the syntax below, an attempt would be made to open a data set named WORK.DATASET:

  ```sas
  disp = open("DATASET","r");
  ```

- To save memory, use a length statement to define temporary SCL variables (those that do not appear on the DISPLAY window). The default is created as a character variable, 200 characters long.

- When possible, use SCL functions to achieve the desired outcome rather than submitting SAS source statements in a SUBMIT block. Instead of using a SUBMIT block to issue a FILENAME statement, use the following SCL statement to accomplish the same task:

  ```sas
  rc = filename('GOTU', 'USER_RW.GOTU792');
  ```

- When using a SUBMIT block to access SQL, use the SQL argument on the SUBMIT statement rather than a SUBMIT block that submits the SQL procedure. The key here is that SUBMIT SQL processes the SQL task, which is more efficient than invoking PROC SQL.

- Unless you need to modify attributes "on the fly," use the ATTR or GATTR windows in SAS/AF and Option 5 from the Modification Menu when in PROC FSEDIT, "Modification of General Parameters." Using this procedure includes attributes from PROTECT to window sizing.

- Use field types in the attribute window to decrease the amount of SCL required. If you specify a field type of INPUT, when the user types a "?" in the field, a selection list is automatically displayed listing members of the SAS data library available for use.

- Use CALL EXECMDI to issue global commands rather than a SUBMIT block with a DM statement.

- Using routines is more efficient than submitting PROC statements. For example, it is better to use CALL FSEDIT than PROC FSEDIT. CALL DISPLAY, CBT, FSVIEW, FSLIST and LETTER are a few of the routines available.

- When using CALL DISPLAY, consider using parameters to pass values between windows rather than CALL SYMPUT and CALL SYMGET. Parameter passing should be used when you want to pass information directly to the next SASA/F window in your application. CALL SYMPUT and SYMGET should be used when you want values to be available to any subsequent windows, not just one specific entry.

- If you will be fetching a lot of data from a SAS file, using a FETCH/SBS function in the INIT section will make subsequent FETCH functions perform much more quickly. The argument to the FETCH/SBS function should be a value greater than the number of observations in the SAS data set—so you are effectively fetching past end-of-file.

- Using the DATA/LIST function is more efficient for displaying selection lists than building an extended table if the values you want to present are data set variable values.

- If you have created indexes on a data set in Release 6.06, using the SCL WHERE function (or WHERE= data set option on the OPEN function) is more efficient than submitting a DATA step to subset the data.

- In Release 6.06 you can use the FIELD function to determine if fields on the DISPLAY window have been modified. Using an array as an argument to the FIELD function in this case is more efficient that using the MODIFIED function for each field.

- When possible, limit the number of SCL variables that your application references. When X and Y are window variables, instead of

  ```sas
  /* SCL code precedes to this */
  TROVE1 = 'x=';
  TROVE2 = 'y=';
  TROVE3 = 'z=';
  RC = WHERE(RS, X + '1' || Y + '1' || Z + '1');
  /* more SCL code */
  ```

  you

  ```sas
  /* SCL code precedes to this */
  RC = WHERE(RS, X + '1' || Y + '1' || Z + '1');</sas>

  " /* more SCL code */"
• Processing time is faster when you use an array as the argument to the IN operator as opposed to using a list of values.

```sas
ARRAY INDEX(100) I=1..100;
T = 'PART' IN INDEX;
```

rather than

```sas
T = 'PART' IN ('101', '102', '103', ...);
```

• If you are reading a data set but modifying only a limited number of variables, using CALL PUTVAR and CALL GETVAR are more efficient than using CALL SET. With CALL SET, every time an observation is fetched, all variable values in the data set are read into the SQL Data Vector.

• When executing commands in a PROC FSVIEW session, design your formula so that users issue commands only once, rather than each time they update an observation or scroll. For example, if you wanted to identify a customized help entry, you could create a variable called HELPVAR. The formula for this variable could be defined:

```sas
HELPVAR=HELPVAR;
IF TYP = 'TYP' THEN DO;
   CALL PRINT('GATTA HELP.CAT,MT,HELP');
   TYP=1;
END;
```

Since the condition would be met only once, the command would only be issued one time. The DROP command could be used to drop the variable, HELPVAR, from the display. Even without the variable displayed, the SQL would still execute.

• Rather than opening and closing data sets multiple times, when possible open them in FSEINIT (SAS/FSP software) or INIT (SAS/AF software) and close them in FSETERM (SAS/FSP software) or TERM (SAS/AF software).

### MAINTENANCE CONCERNS

Here are some techniques that may save you time in the long run because they will minimize questions from your users or save you maintenance time. They are as follows:

• Comment your code. Returning to review your logic later will be easier if you have short notes, but keep these comments to a minimum. Comments within a SUBMIT block are considered strings and are included in the calculation of code size.

• Provide customized help for your users to access.

• Always provide a means of exiting the program, (and, of course, the application).

• You may find that using customized key settings, such as PMENUS or selection boxes, is an easier method of helping your users move through the application; that way they do not have to remember default key settings. This is also a good way to limit the choices a user can make on the screen, and thus the amount of coding you must do.

• A technique that many application developers use to minimize control of program flow in Release 6.05 is to specify that no command line appear on the SAS/AF entry. This can be achieved by specifying BANNER NONE on the GATTA window. However, this may also mean that messages that had been sent to the MESSAGE LINE will no longer be displayed for your user. You can still provide those messages to your users in one of two ways:

  • Create a dummy PMENU:

    ```sas
    PROC FSPM ENUS catalog=myrop.catalog;
    PMENU APP;
    ITEM 'HELP';
    END;
    ```

    Specify the menu name, in this case APP, in the COMMAND MENU field on the GATTA window. Also specify BANNER NONE on the GATTA window. Specify PMENU ON (by either using CALL EXECCMD or setting it in your AUTOEXEC file).

  • Specify CONTROL ALWAYS; in the INIT label. Set a field equal to _MSG_. Note that in order for the user to see the message text, the field must be displayed, but you can specify that the PAD character be blank. This would result in the field appearing only when there is a value in it.

• Review your source code. As you become more familiar with SCL, you may find that functions or methods of coding you originally overlooked will save either processing time or disk space.

These are just a few of the tips that you can use to assist you in creating the most efficient Screen Control Language programs. As you write SCL programs, you will find others that work especially well given the needs of your users and your application environment.

SAS, SAS/AF, and SAS/FSP are registered trademarks of SAS Institute Inc., Cary, NC, USA.