Efficiency Techniques for SAS/AF® Software Applications

Annette Harris, SAS Institute Inc., Cary, NC

There are several techniques that save you creation and execution time with SAS/AF® software applications.

THE INCLUDE COMMAND

The type of terminal on which the application will be executed is important in any full-screen application. In Version 5 SAS/AF software, the terminal configuration is stored with the screen. This means that you must be careful that the terminal on which you do the development work has the same configuration as the terminal your users will be accessing. Should a problem arise, there is a method of restoring the screens without manually having to re-create your entire application. In Version 5, one relatively quick means of screen retrieval is to write the screen to an external file. Then, edit a new entry in the catalog, and use the INCLUDE command to bring the information from the external file. This enables you to restore the display screen and SAS® code, but the attributes for the screen need to be respecified. Version 6 SAS/AF software addresses this problem by enabling you to modify the size of display windows.

FAST BRANCHING

Systematically, the number of DISPLAY procedure statements used significantly affects the performance of your Version 5 SAS/AF application. For this reason, the >>> and >> branch indicators are beneficial because they can reduce the number of PROC DISPLAY invocations. The >>> indicator designates that an unconditional fast branch should be executed to the screen whose name is provided following the indicator. Unconditional fast branching means that any SAS code appearing on the screen is not executed. The >> branch is a conditional branch. The branch is executed only if the field specified has a non-blank value. It is possible to suppress unconditional branches by using the ### macro (a custom screen control macro) and ## conditional statement indicators. The following example conditionally branches based upon the value supplied by the user:

```
   Please type in the name of the report you wish to see:

        &_____          &._____________________

          Valid report names are: MONTHLY and YEARLY
-------------------------------------------------------------------------
### move
%macro move;
 %if &_dcall=INITIAL %then %do;          /* Check for initial      */
    %let field1=;                        /* screen display and     */
    %let _dmsg=;                         /* reset fields.          */
    %let _derron=;
 %end;
 %else %do;                              /* If not, check for valid*/
 %if &sysjobid=%str(validid) %then %do;  /* ID and field value.    */
     %if %str(&field1)=%str(MONTHLY) %then %do;
       %let field2=monthly.program;      /* If valid, set field2.  */
       %let _dmsg=;
       %let _derroff=;
     %end;
     %else %if %str(&field1)=%str(YEARLY) %then %do;
       %let field2=yearly.program;
       %let _dmsg=;
       %let _derroff=;
     %end;
     %else %if %str(&field1)¬=%str(YEARLY) and %str(&field1)¬=
        %str(MONTHLY) %then %do;
       %let _dmsg=Please enter a valid report value.;
       %let _derron=field1;
     %end;
 %end;
 %else %do;                              /* Else give error message.*/
  %let _dmsg=You are not authorized for this info.
             Press the END key.;
  %let field2=main.menu;
 %end;
%end;
%mend move;
###

##field2
>>> &field2 /* If field2 not blank, branch to screen specified.*/
```

Note: both fields have associated macro variables specified that are the same name as the user field. FIELD2 has protect and nondisplay attributes set.

CUSTOM KEY SETTINGS

Another technique to save execution time is to modify the keys settings for the end user. In Version 5, reserved names are used to identify the keys for particular screens. For example, DISPLAY.KEYS is the name of the entry used to determine the keys used by the DISPLAY procedure. In Version 6, you can choose the name of the KEYS entry, but you must specify that entry name in the general attribute panel. You can also use the Screen Control Language (SCL) to define commands that are not SAS commands. If you use SCL commands, issue the VERIFY OFF command from the command line of the KEYS entry so that the SAS System does not check for valid SAS commands. In either version, a KEYS entry can be created in the catalog by issuing the command `EDIT` *`name`*`.KEYS` on the command line of the catalog directory screen. The search sequence for keys is as follows:

Current Catalog SASUSER.PROFILE WORK.PROFILE SASHELP	Version 5
Current Catalog SASUSER.PROFILE SASUTL	Version 6

CHECKPOINT RECORDS

Execution flow may be controlled with checkpoint records. Checkpoint records written by PROC DISPLAY (Version 5) or the AF command (Version 6) are stored in the profile catalog. These records maintain information used when the RECALL command is executed or when a PROC DISPLAY statement is executed and a four-level name is not supplied. Checkpoint records also store course-related data (such as number of questions seen and number of tries used per question) for computer-based training (CBT) courses.

In Version 5, there are four types of checkpoint records: CBTGO, CBTSAVE, PGMSAVE, and TRANSAVE. Although these appear as entries in a catalog, they cannot be browsed or edited. The

CBTSAVE checkpoint record contains information about the last frame seen (maximum number of questions seen, number of tries per question, and so on). The CBTGO checkpoint record is written when the SAVE command is issued in a CBT course. The CBTGO checkpoint record overrides the CATALOG= specification in the PROC DISPLAY statement, enabling the student to return to the point in the course last exited rather than starting at the beginning of the course each time. The PGMSAVE checkpoint record is accessed when the RECALL command is executed while in PROC DISPLAY on a program screen. The RECALL command returns values previously entered by the user when there is no associated macro variable for the field. The PGMSAVE checkpoint record is stored in a catalog named WORK.AFCKPT. The TRANSAVE checkpoint record is referenced when the PROC DISPLAY statement is executed but a four-level name is not supplied. The TRANSAVE checkpoint record is stored in the profile library and contains the libref, catalog name, entry name, and entry type of the screen.

What happens when the CANCEL command is issued, and how does this fit into the Version 5 checkpoint concept? SAS/AF software maintains a stack that is accessed when the CANCEL command is issued. This stack contains PROGRAM screens and the screen previously displayed (a MENU, CBT, HELP, or PROGRAM screen). If the first screen displayed in an application is a PROGRAM screen, the entry name from the TRANSAVE checkpoint record is added to the stack in the event a CANCEL command is issued from the first screen.

In Version 6, the SASUSER.PROFILE catalog is used to store information on what screen should be displayed if the CATALOG= option is not present in the AF command. This catalog is accessed when the SAVE and RECALL commands are issued. The entry AF.AFGO in the SASUSER.PROFILE catalog stores the four-level name of the screen that should be displayed when the AF command is executed and the CATALOG= option is absent. The name of the entry accessed when the SAVE and RECALL commands are issued is the same as the screen from which those commands are executed. The entry type is AFPGM. When the CANCEL command is executed, you return to the screen previously displayed. The CHECK= option in the AF command determines whether or not the name of the last MENU or CBT screen displayed is stored when you exit the application.

In both versions, the concept behind the checkpoints and the stacks is that the user should never unintentionally exit a SAS/AF software application. Knowledge of how and when checkpoints are used will enable you to ensure a controlled environment for the end user.

TOKENIZATION

In Version 5, understanding how SAS/AF software tokenizes the code below the dashed line on a program screen saves you time in the creation and debugging of applications. Each line is read in and split into tokens with the delimiter being a blank. There is an implied delimiter at the beginning and end of each line. Each token is examined to see if it contains any special push indicators. If ##, ¬##, or any of the other push indicators are encountered, a look-up is executed to verify that the character string following the indicator is the name of a user field. Then, switches are set to control the subsequent action that is taken (whether the code is pushed or not). At this point, the special indicators are blanked out because they are not part of the pushed code. If no special indicators are found, a check is done to see if the token starts with an ampersand (&) followed by a valid user field name. If an ampersand is encountered, a recursive algorithm is executed that breaks the tokens apart and does substitution on them. If the resulting line is not blank, another routine is entered that breaks the line into pieces for submission to the SAS word scanner. The token following the user field is moved left to squeeze out any trailing blanks. Subsequent tokens can remain in their original position, and therefore blank spaces can occur within a line.

In the following example, the user fields, USRID and JOBNAME, have the non-display attribute specified so that when the user sees the screen, these fields do not appear. USRID has an associated macro variable, SYSJOBID, specified. An associated macro variable, JOB, is provided for the user field JOBNAME. When the user enters the information on the screen and presses the END key, the data are substituted into the JCL and written to the external file referenced by the fileref OUT.

```
&USRID_______

              PLEASE ENTER THE FOLLOWING:

           LOCATION:  &LOC_________
          YOUR NAME:  &NAME____________________
           PRIORITY:  &P
         JOB NUMBER:  &JB_____

&JOBNAME____________________
          PRESS THE END KEY WHEN YOU ARE FINISHED
------------------------------------------------------------------
=== out  /* write to external file pointed to by fileref out */
//&USRID&JB JOB (,&LOC,,9999),'&NAME',PRTY=&P,
//            NOTIFY=&USRID,CLASS=A
===
```

This code writes information to the external file referenced by the fileref, out. The contents of that file are:

```
//YOURID1   JOB (,NC,,9999),'  ANNETTE HARRIS',PRTY=1,
//            NOTIFY=YOURID,CLASS=A
```

Because of tokenization, note the blanks preceding the word JOB and the user's name. This occurs because the USRID value was substituted in. The job number value was substituted in and moved left to squeeze out trailing blanks left when the USRID value was not as long as the USRID user field. A blank space is recognized as a delimiter and the characters JOB are encountered. These characters remain in their original location. The same type of tokenization occurs again when the value from field LOC is encountered. A slight modification to the code eliminates the blanks:

```
### fix
%macro fix;
   %let uid=%trim(&sysjobid); /* Trim trailing blanks from sysjobid.*/
   %let jobname=&uid.&jb  JOB;/* Create new macro variable.         */
%mend fix;
###
=== out
//&JOBNAME (,&LOC,,9999),'&NAME',PRTY=&P,
//            NOTIFY=&USRID,CLASS=A
===
```

Note: the SYSJOBID macro variable returns a length 8. The TRIM autocall macro is used to trim trailing blanks from the resultant value that might result in improper tokenization of the line.

The code above will result in the following being written to the file:

```
//YOURID1  JOB (,NC,,9999),'ANNETTE HARRIS',PRTY=1,
//            NOTIFY=YOURID,CLASS=A
```

THE ### MACRO AND SAS/AF MACRO VARIABLES

The ### macro in SAS/AF software enables validation and initialization of values in user fields. It also enables you to send customized messages to the message line of the program screen

your users are viewing. This special macro is executed once just prior to display of the screen and again any time the user presses the ENTER key or any function key. The following macro variables are available for use within the ### macro environment:

_DCALL	_DMSG
_DERRON	_DKEY
_DERROFF	_DKEYDEF
_DALARM	_DLASTC
_DCURSOR	_DCMND

These macro varaibles are documented in Technical Report P-149: *Changes and Enhancements to SAS/AF Software* and Technical Report P-146: *Changes and Enhancements to the Version 5 SAS System*. If you have a segment of code that will be used many times, you can store that code in a macro in an autocall library. To invoke these macros, simply enter `%macroname` or `### macroname` on your program screen below the dashed line. When an autocall macro is used, the MAUTOSOURCE option should be specified. Once compiled, macros that are in the autocall library are not recompiled.

The following example uses the ### macro to execute the DIR command enabling the user to list the variable names and to type in the data set specified:

```
                DIR command program

&  Do you want to see a listing?

Type in the data set name; then press the ENTER key: &dset_________

Press END if you do not want to see a listing.

-------------------------------------------------------------------------
### ck
%macro ck;
  %if &_dcall=initial %them %do;   /* Is this initial display?    */
   %let listvar=;
   %let dset=;                     /* Reset macro variable values.*/
  %end;
 %else %do;
   %let _dcmnd=;
   %if &listvar¬=and &listvar¬=%str( ) %then %do;
     %let _dcmnd=dir &dset;        /* If not initial display,     */
     %let listvar=;                /* and LISTVAR is not blank,   */
   %end;                           /* issue DIR command.          */
   %%lse %do;
     %if &_dcall=end %then %do;    /* otherwise, END.             */
        %let listvar=;
     %end;
   %end;
  %end;
 %mend ck;
 ###
 ppp main.menu
```

Note: the first field on the screen, an ACTION field, has an associated macro variable, LISTVAR. The second field, &DSET, has an associated macro variable that has the same name as the DSET field.

In the example program below, a user field is checked to see if it has a value. If it does not, the user is prompted to provide a value. If the field has a value and the ENTER key is pressed, the END command is executed. Again, user field TESTAVAR has an associated macro variable by the same name.

```
                TEST.PROGRAM


&testavar

-------------------------------------------------------------------------
### test
%macro test;
   %let _dcmnd=;
     %if &_dcall=initial %then %do;   /* Check for initial display.*/
         %let testavar=;
     %end;
   %else %do;    /* If field blank and ENTER pressed, send message.*/
     %if (&testavar=%str( ) or &testavar=) and &_dkey=0 %then %do;
   %let _dmsg=Please type in a value;
     %end;
      %if (&testavar¬=%str( ) and &testavar¬=)
          and &_dkey=0 %then %do;
      %let _dmsg=; /*If field non-blank and ENTER pressed, issue END.*/
      %let _dcmnd=end;
     %end;
   %end;
 %mend test;
 ###
```

The examples in this paper illustrate some of the capabilities of SAS/AF software. Considerations of hardware, end-user knowledge, flexibility requirements, and environmental constraints are dealt with more positively and efficiently with the implementation of various features of SAS/AF software in your customized full-screen application.