

SAS/AF® Software Tutorial

Phil Busby, SAS Institute Inc.

This tutorial covers advanced techniques for SAS programmers to create full-screen, menu-driven application systems using SAS/AF software. The following topics are included:

- Using SAS AUTOEXEC to go directly to the first screen of your application.
- Writing a **###** MACRO to control the display of a program screen.
- Routing program screen output with userfield value substitution to an external file using the **===** FILEREF option.
- Setting up a CBT automation sequence using the AUTO option.
- Displaying a graph and a CBT frame simultaneously with the GRAPH option.
- Tracking the answers to your CBT multiple-choice questions with the QUIZ option.

Section 1. The SAS AUTOEXEC facility for custom initialization.

With the SAS AUTOEXEC facility of the SAS Display Manager System, you can configure the SAS session for your user, and put the user onto the main menu of your SAS/AF application system. The user needs to re-member one command to enter on the host operating system command line. That command, whatever you choose it to be, invokes the SAS System. When display manager gets initialized, it does the AUTOEXEC commands, one at a time, just as if you were keying them in for the user. The last AUTOEXEC command should be a SUBMIT command, which feeds into the SAS word scanner some SAS statements for execution. Include a PROC DISPLAY statement for the main menu, and that statement's execution will take the user into your custom application system.

Here is an example for an in-house training course. The AUTOEXEC commands change the default function key values so that the user can jump from the procedure output screen back into the middle of the CBT lesson simply by pressing a function key.

```
PROC 0 GO +
  TRACE
  IF &TRACE = &STR(TRACE) THEN -
    CONTROL MSG CONLIST PROMPT
  ELSE -
    CONTROL NOMSG PROMPT
  FREE P(CBT,SASUSER,SASEXEC)
  ALLOC F(SASUSER) UNIT(VIO) NEW TRACKS SP(5 1) DELETE
  ALLOC DA('SASPPB.CBT.DEMO') FILE(CBT) SHR
  ALLOC F(SASEXEC) +
    NEW TRACKS SP(5 1) UNIT(VIO) DELETE
  /*
  /* SET UP AUTOEXEC
  /*
  OPENFILE SASEXEC OUTPUT
  SET &SASEXEC= &STR(PGM;)
  PUTFILE SASEXEC
  SET &SASEXEC= +
    &STR(%LET DMSPPK18=&STR(SUBMIT 'PROC DISPLAY;RUN;');)
```

(continued on next screen)

(continued from previous screen)

```
PUTFILE SASEXEC
SET &SASEXEC= +
  &STR(%LET DMSOPK18=&STR(SUSPEND;SUBMIT 'PROC DISPLAY;RUN;';END;))
PUTFILE SASEXEC
SET &SASEXEC= +
  &STR(%LET DMSLPK18=&STR(PGM;SUBMIT 'PROC DISPLAY;RUN;'))
PUTFILE SASEXEC
SET &SASEXEC= +
  &STR(SUBMIT 'PROC DISPLAY C=CBT.MAINDEM.TITLE.CBT; RUN;')
PUTFILE SASEXEC
CLOSEFILE SASEXEC
/*
/* INVOKE THE SAS SYSTEM TAKE USER DIRECTLY TO CBT TITLE SCREEN.
/*
SET OPTIONS=&STR(NONOTES NOSOURCE NONUMS NODATE NONUMBER +
  NONACROGEN NOSYMBOLGEN NOSOURCE2 NAUTOSOURCE MCOMPILE SYSPARM='CBT')
SASTEST SHARE + OPTIONS('MACRO DQUOTE 'Q') +
  AUTOS('SASPPB.DEMO.CBTAUTOS') + %GO %DEBUG %TESTRAM %TRACE
FREE P(CBT,SASUSER,SASEXEC)
```

Section 2. The **###** macro for custom control of program screens.

With a **###** macro you can do consistency checks, generate special messages, generate commands, and set indicator userfields to tailor the execution of a program screen to your custom requirements. The macro is executed once just before the screen is displayed and once each time the user presses ENTER or a function key. The name of the macro goes after the three pound signs on the first line following the dashed line of a program screen. You can then optionally define the macro, and a second **###** line marks the end of the macro and the start of other program screen data, such as SAS source statements. Here is a typical **###** macro that does

a consistency check to allow a user in the payroll department to run a salary report on any department except payroll:

```
Please enter the employee department:  &dept____
Please enter the type of report wanted:  &reptype
Thank you.  Press END to see the report.
```

(continued on next screen)

```
(continued from previous screen)

*** CHECKIT
$MACRO CHECKIT ;
  IF &_DCALL = INITIAL $THEN $DO ;
    $LET DEPT= ; $LET REPTYPE= ;
  $END ;
  $ELSE $DO ;
    $IF $QUOTE(&SUPERQ(DEPT)) = PAYROLL AND
      $QUOTE(&SUPERQ(REPTYPE)) = SALARY
    $THEN $DO ;
      $LET _DERRON= DEPT REPTYPE ;
      $LET _DMSG= Salary report for payroll dept. prohibited. ;
    $END ;
    $ELSE $DO ;
      $LET _DERRON= ; $LET _DMSG= ;
    $END ;
  $END ;
$MEND CHECKIT ;
***
DATA TEMP ;
  SET MASTER.EMPLOYEE ; IF DEP = '&DEPT' ;
DATA _NULL_ ;
  SET TEMP ;
/* Report generation statements ... */
RUN ;
```

Please look carefully at the line in the DATA step above:

```
IF DEP = '&DEPT' ;
```

We assume that DEP is the name of the variable in MASTER.EMPLOYEE that has the employee's department code. We want to compare it to the department code requested by the user for the report; since the variable is character, a character literal is used. You might suppose that the subsetting IF statement should read:

```
IF DEP = &DEPT ;
```

in order to compare the variable DEP with the value of the macro variable DEPT. &DEPT is not a reference to a macro variable here. It is a reference to a userfield called DEPT. Suppose the user types SALES in the DEPT userfield.

PROC DISPLAY substitutes SALES where &DEPT is in the SAS code below the line, and what gets pushed to the SAS word scanner is

```
IF DEP = SALES;
```

and SALES is interpreted as a variable rather than a value. If you want to push code that contains references to SAS macro variables, put two ampersands in front of the macro variable name. PROC DISPLAY will realize that the name following the double ampersand is a SAS macro variable, not a userfield reference, and it will delete one of the ampersands for you and push the rest.

You might also notice the reference to _DCALL in the *** MACRO above. This is an automatic macro variable available to you within your *** macro to make your initialization and termination routines easy to code. When _DCALL = INITIAL, the user has not seen the screen yet. If the user enters the END or CANCEL command, then _DCALL will be equal to END or CANCEL; otherwise, _DCALL equals PARSE.

There are nine other automatic macro variables you may find useful:

_DERRON	contains the names of macro variables whose userfields were found by PROC DISPLAY to be in error. You can add others that you find in error.
_DERROFF	lets you turn off the error status of userfields.
_DKEY	contains the function key number that the user pressed.
_DKEYDEF	contains that function key's command equivalent.
_DMSG	lets you put a message on the message line.
_DCURSOR	tells you what field the user put the cursor on.
_DALARM	lets you sound the alarm.
_DLASTC	gives you the last command entered so you can check for your own special commands.
_DCMND	lets you feed in a command for execution.

Section 3. The == fileref for putting the user's values in a file.

This feature of SAS/AF program screens allows you to push lines to an external file rather than to the SAS word scanner. You can achieve the

same result with a DATA step containing FILE and PUT statements, but the === fileref saves you the DATA step overhead.

Following the dashed line of a program screen, and after any ### macro and >> conditional branch lines, put three equals signs in the first three columns of the line, and follow them with the name of the fileref to which your external file is assigned. The lines following the === fileref line, up to but not including another === line, will be written to the external file when the END command is executed by PROC DISPLAY for the program screen. The lines to be written are first scanned for userfield references and conditional push indicators.

The following example shows how you can let the user submit a batch SAS job from a SAS/AF screen.

```

Full Screen Catalog Utility

Please enter the screen you want printed:

SLIB_____ SLEN_____ SOBJ_____ STYPE_____

Please enter the SAS data library containing the screen:

SDATASET_____

Thank you. Press END to submit the batch print job.
-----

```

(continued on next screen)

```

(continued from previous screen)

=== BATCHQ
//PPBRATCH JOB 'SAS.PPB',CLASS=X,NOTIFY=SASPPB,TIME=(001,10)
/*JOBPARM PETCH
// EXEC SASSTEST
//SASUTL DD DISP=SHR,DSN=SAS.TEST.UTLLIB
//SLIB DD DISP=SHR,DSN=SDATASET
//SYSIN DD *
X ALLOC FI(SLIB) DA(SDATASET) SHR;
/* THIS JOB PRINTS THE SPECIFIED SAS/AF SCREEN */
PROC DISPLAY C=SLIB.SLEN.SOBJ.STYPE;
RUN;
/*
***

/* NOTE: SASPPB.AF.OUT HAS BEEN ASSIGNED TO FILEREF BATCHQ */

X SUBMIT 'SASPPB.AF.OUT';
PROC DISPLAY;
RUN;

```

If you are not going to submit any statements to the SAS word scanner, but just write them to an external file, then you can branch your user directly to another screen after the lines are written using >>> like this:

```

=== BATCHQ
//PPBRATCH JOB 'SAS.PPB',CLASS=X,NOTIFY=SASPPB,TIME=(001,10)
/*JOBPARM PETCH
// EXEC SASSTEST
//SASUTL DD DISP=SHR,DSN=SAS.TEST.UTLLIB
//SLIB DD DISP=SHR,DSN=SDATASET
//SYSIN DD *
X ALLOC FI(SLIB) DA(SDATASET) SHR;
/* THIS JOB PRINTS THE SPECIFIED SAS/AF SCREEN */
PROC DISPLAY C=SLIB.SLEN.SOBJ.STYPE;
RUN;
/*
===
>>> ANOTHER PROGRAM

```

Section 4. The AUTO option for playing a moving picture.

You can make your application system come alive by displaying a set of screens one after the other without requiring the user to press a key to go to the next screen. You specify this feature by putting the keyword AUTO on the first line of your set of frames in a CBT object. The CBT object looks like a HELP screen; you can use colors and attributes anywhere on displayed text lines. When PROC DISPLAY processes a CBT object, it looks for dashed lines or lines beginning with a question mark to know where each frame begins. When a frame begins with a question mark, the procedure scans the rest of the line for special instructions. The AUTO option is one of the instructions you can specify to modify the way the frames are displayed. It tells PROC DISPLAY to display frames without waiting for a keystroke. You can also say how fast to display the frames, like this:

? AUTO=5

This specification tells the procedure to display five frames per second. If you leave off the "=5" and put just the word AUTO, then the frames are displayed as fast as possible. If your CBT object has multiple-choice questions in it, then automation is suppressed; in that case, you should put the automation sequence in a separate CBT object and hook them together by making the first CBT object's child screen the name of the second CBT object so that when the last frame of the first object is displayed, the procedure branches to the first frame of the second object.

Since CBT frames are made up of lines and each line is an entire field on the terminal screen, your automation sequence can go faster if fewer lines change when going from one frame to the next.

Section 5. The GRAPH option for displaying graphics and text together.

The GRAPH option tells PROC DISPLAY to fetch a graph from a graphics catalog, display the graph, and then display a CBT frame on top of the graph. The result is graphics and text displayed simultaneously. You set this up by putting the GRAPH= option on the ? line starting a CBT frame. Here is a basic example:

```
? GRAPH=LIBREF.GCAT.PIC/ERASE
```

The ERASE keyword following the slash tells the system to erase any previously displayed graphs before showing PIC. If you leave off the ERASE option, then the picture is displayed along with graphs from prior frames. This means that you can put up a graph and it will stay up until you say to erase it. If you want to erase all previous graphs without displaying any new graphs, tell the procedure, like this:

```
? GRAPH=ERASE.ERASE.ERASE/ERASE
```

The graph normally goes into the default window starting in column 6, and going across to column 75, and starting in the second text row and going down to row 19, leaving 11 lines at the bottom of your 30-line terminal screen for text below the graph. If you want the graph to go into some other part of the screen, you can give the coordinates like this:

```
? GRAPH=(6,75,4,12)LIBREF.GCAT.PIC/ERASE
```

When the default is not used, you must give all four coordinates in this order: left column, right column, top row, bottom row. The graphics catalog may be in the same SAS data library that the CBT object is in, or it may be in a different catalog. The most important point to remember when using this feature is that you must put blank lines in your CBT frame where the graph will go.

Section 6. The QUIZ option for tracking CBT question responses.

This feature is used by CBT course authors to see how students respond to multiple-choice questions. The builder sets up a SAS data set for PROC DISPLAY to write tracking data into,

and PROC DISPLAY gets the name of your tracking data set from the QUIZ= option on the first ? line starting a CBT object:

```
? QUIZ=LIBREF.SASDS
```

The specification above says that this is the start of a new frame (by the question mark in column 1) and that this is a multiple-choice question frame with two tries allowed (by the digit 2 in column 2) and that tracking data should be written to LIBREF.SASDS (by the QUIZ= specification). The system will open the SAS data set for update and append observations to it. You get one observation in the tracking data set per QUIZ frame. A QUIZ frame is specified by typing the keyword QUIZ anywhere on the ? line starting a question frame. In other words, each frame is tracked separately, and only the question frames marked with the keyword QUIZ are tracked.

Each observation written to the tracking data set contains values for seven variables:

LIBREF	the SAS data library assigned
CATNAME	the catalog name that the CBT course is in
OBJNAME	the CBT object name
FRAME	the sequence number of the QUIZ frame being tracked
TRIES	how many tries the student used to answer the question
MAXTRIES	the maximum number of tries allowed
SCORE	1 = Answered correctly, 2 = frame not seen by student, 0 = not answered correctly but some tries remaining, -1 = not answered correctly and tries used up.

The first three variables are eight characters long. The first four variables uniquely identify the question frame. The last four are numeric variables.

Tracking data are not written out until the student ends the course, either by pressing ENTER on the last frame or by pressing END to exit. If a student branches to another screen from the middle of a CBT object and later resumes the course where he left off, the tracking data are saved in the CBT checkpoint record and not written out until the course is ended. Perhaps the student branches out of your course and does not resume. You will know that the student at least started the course because a "frame zero" record is written to the

tracking data set whenever a branch out of the CBT object is taken. A frame zero observation contains the value of zero for all numeric variables.

In addition, it is possible for the student to press END on frame 7 of a 12-frame CBT object, and you will get no tracking data for frames 8-12 unless the student jumped ahead in the lesson to see them.

Summary of SAS/AF advanced features:

- AUTOEXEC=
- ### MACRO
- === FILEREF
- CBT AUTO=
- CBT GRAPH=
- CBT QUIZ=

These are some of the features available in SAS/AF software to make designing full-screen, menu-driven systems easy. Your users can go right into your application system smoothly with an AUTOEXEC file that submits a PROC DISPLAY statement for them. You can add a ### MACRO to your program screen and control its execution. The === FILEREF lets you route the code below the dashed line on a program screen to an external file. You can branch to CBT screens with the AUTO option or GRAPH option and make your system come alive with moving pictures and graphics, and the QUIZ option allows tracking of students' answers to CBT questions.