SAS/AF software is a new tool that allows SAS programmers to create full screen menu application systems for non-technical end users. This tutorial covers the preparation of Help, Menu, CBT, and Program screens using PROC BUILD. The SAS programmer can easily tie these screens together into a complete information center tool that end users may access through PROC DISPLAY.

PROC BUILD, which gives you the powerful Display Products Editor to facilitate screen creation, will be demonstrated, and many useful editing commands will be illustrated. You will see how to set up branching to other screens in response to the user’s menu selection. At the heart of SAS/AF software are program screens, consisting of (optional) displayed text that the user sees and SAS statements that are submitted to the SAS System. The technique of substituting user-keyed values into the SAS source statements will be discussed. The use of SAS macro variables to create menus dynamically will be illustrated. Also, you will see how the AUTOEXEC facility of the SAS Display Manager can be used to initialize your full-screen menu application automatically.

This tutorial will demonstrate how to use SAS/AF software to create menu-driven, full-screen applications for end users who may know little about data processing. Your end users will be able to select items from a menu that you can set up for them. They can substitute values into SAS application programs and run those programs without ever seeing any SAS code.

When your user selects an item on a menu, you can cause the system to go to one of four types of screens. You can go from one menu to another menu, to a help screen, to a CBT screen, or to a program screen. These screens are easily tied together into an integral, customized application system using SAS/AF software.

SAS/AF software consists of four procedures: BUILD, DISPLAY, CPORT, and CIMPORT. The last two are used to move your catalog of screens from one system to another. PROC DISPLAY processes the screens and displays them to the end user. This tutorial will focus on PROC BUILD, which is used to create and edit the screens processed by PROC DISPLAY.

I will start by explaining the way your screens are stored. Both PROC BUILD and PROC DISPLAY have a required parameter, the catalog specification, of the form CATALOG=libref.member.object.type. PROC BUILD needs only libref.member. The libref is the external data set reference that most IBM users call the DD. You must assign this logical name to the physical data set where you want your screens to be stored prior to executing a SAS/AF procedure. The libref you specify may also contain regular SAS data sets. They are kept separate by the system through the second part of the catalog specification, the memname, which is any unique identifier up to six characters long that you may choose. The memname specifies a group of screens within the libref.

The third part of the catalog specification is the screen name, also called the object name. Let me point out here that “screen” and “object” are the same thing in SAS/AF software; the terms refer to a collection of lines that are displayed together in a full-screen environment. An object may consist of more lines than will fit on the CRT; the user may use the scroll commands FORWARD and BACKWARD to see them. The fourth part of the catalog specification is the object type, which must be HELP, MENU, PROGRAM, CBT, LIST, SYSTEM, or FORM. The last type, FORM, is used only by SAS/FSP procedures for printing to different devices. SYSTEM screens are for internal use by SAS full-screen procedures.

For PROC BUILD, all you need on the catalog specification is the libref and the member because PROC BUILD works with the whole catalog. PROC DISPLAY works with one object at a time, so the full four-level catalog specification is needed for PROC DISPLAY, or just three levels if the object name is unique in that catalog.

When you execute "PROC BUILD C=lib.mem; RUN;" you will see the catalog display, showing each object in the catalog, one object per line. To choose an object to work with, use the EDIT command. When you key in "EDIT A.HELP" the procedure will display the help object A for update if it exists; otherwise, it assumes that you want to create a new help object called A.

When you are editing a SAS/AF screen, you have all of the editing commands of the Display Products Editor available, and you can enter "NUMBER ON" for entering line commands. If your terminal has extended highlighting capabilities, you can use them anywhere in the data area to make lines or parts of lines a different color or a different attribute such as reverse video, blinking, or underline.
After you have keyed the lines of help text, press the END key to save your object in the catalog, and you are returned to the list of catalog entries.

You can select an existing object for editing either by keying the EDIT command followed by the object name or by placing the letter "S" (for "select") to the left of its name in the catalog screen. The name of an object or its description can be changed right on the catalog screen by keying the letter "R" (for "rename") to the left of the object name. To delete an object from the catalog, you can enter "DELETE objectname.type" on the command line.

MENU objects are created in the same way that HELP objects are created. Enter "EDIT A.MENU" on the command line, and you are given a blank slate for keying in what you want the menu to look like.

To make the connection between the item on the menu and the object you want to transfer to when the user selects that item, use the ATTR command, placing the cursor where you want to split your screen. The attribute panel for the menu will be displayed in the bottom part of the split screen. You may have up to sixty items on one menu; if you need more, divide them into groups of sixty or fewer items each, and let your user go from a main menu to one of the group menus. Note that the transfer object for a menu item may be another menu, a help object, a CBT object, or a program object.

Now let's look at the menu attribute panel, showing the menu's parent and children. The library and member names for each child screen are initialized to an asterisk. This asterisk notation means that the specified object exists in the current catalog. If you want to transfer the user to an object in a different catalog, then you can enter the catalog and member names in place of the asterisks; otherwise the asterisk notation saves you the trouble of keying in the library and member names when they do not change.

The user selects a menu item by keying the integer corresponding to the menu item on the command line in PROC DISPLAY. You can allow the user two additional ways to choose a menu item: either by a one- or two-character code or by a PF key. These additional selection methods, when specified on the menu attribute panel, override any other meaning they might have. For example, you could say that item one on your menu can be selected by keying the letter "K" or by pressing PF 20, but the letter "K" will no longer be recognized as an abbreviation for the "KEYS" command; instead, it will mean that the user has selected item one. Similarly, PF 20 will also mean that the user selects item one and not the default meaning of PF 20, which is the FORWARD command. Of course, these overrides are in effect only at the time that the menu is being displayed. If you customize a menu in this way, you might want to provide a help screen for new users to see when they enter the HELP command on that menu. You can provide a HELP object pertaining to an entire SAS/AF screen via the HELPNAME command. Suppose you want your users to see HELP when they ask for help on A.MENU. You set up that link when you key "HELPNAME A.HELP" on the command line while editing A.MENU. If you want to find out the name of the currently defined help screen for an object, just enter the HELPNAME command without any operands.

Now let's discuss the parent screen specification. When a HELP, CBT, or menu object is being displayed and the user enters the END command, then the parent screen is displayed. If there is no parent specified, then the procedure terminates. The exception to this rule is when a help or menu screen is displayed in response to a HELP command. After the user asks for help and sees the help information, the END command will return the user to the screen that was displayed when help was asked for, not to the help or menu screen's parent. In general, if you have a main menu and from it the user can go to a submenu, then the submenu's parent should be the main menu.

That is how simple it is to build a menu with SAS/AF software. Now let's create a PROGRAM object with fill-in-the-blank fields. You enter the EDIT command for the name and type of the object you want to create, and a blank edit area is presented where you can enter the lines that make up the program object. The program object has two parts: the top part is what the user sees, and the bottom part is what gets submitted to the SAS System when the user enters END. The two parts are separated by a dashed line. A fast way to get a dashed line is to key "FILL" on the command line and place the cursor in column one of the line you want filled with dashes before pressing ENTER. The dashed line is required even if nothing is to be displayed to the user. For example, suppose you have a menu item labeled "EXIT" that transfers to EXIT.PROGRAM. When the user chooses EXIT, you want to end the SAS session by submitting an ENDSAS statement. EXIT.PROGRAM may consist of only two lines: a dashed line followed by a line with "ENDSAS;" on it.

If you have an existing SAS program in an external file that you want to convert to a SAS/AF Program screen, it is not necessary to key the statements in all over again; you can assign a logicalname to the external file and pull in the source code with the INCLUDE command and specify where you want it to go with an A (After) or B (Before) line command. Also, if you have one screen and you want to make a copy of it, use the COPY command on the
command line to bring in all of the lines from the copied screen into the screen you are editing.

The connection between what the user sees and the SAS code that is submitted is achieved by your userfields. A userfield is an area on the screen that you usually want to be unprotected for the user to key in some information for you. PROC BUILD recognizes an ampersand as the start of a userfield if it is preceded by a space or if it is in column one of a line. You can give each userfield a name by following the ampersand with the userfield name, and you can extend the length of the field by following the name with trailing underscores. For example, you might have the following in your program object above the dashed line:

Hello user.
Please enter your favorite color: &COLOR____

What the user actually sees will look like this:

Hello user.
Please enter your favorite color:

This example contains one userfield called COLOR where up to ten characters may be entered. In the bottom half of your program object, below the dashed line, you may have some SAS code where the value of the userfield COLOR is needed:

DATA _NULL_;
  PUT 'Hello. ';
  PUT 'Your favorite color is &COLOR';
RUN;
PROC DISPLAY will insert the value of the userfield COLOR into the quoted string before submitting the PUT statement to the SAS System.

Now, a straightforward substitution of the user-keyed value into the SAS code may be insufficient for your application. What if you want the value to be converted using an informat or to be uppercased? You can specify such features on the attribute panel for the userfield. Enter "ATTR" on the command line while editing the program object, and PROC BUILD scans the user display area, looking for userfields, and it displays to the builder a panel of attribute information for each userfield. Notice that PROC BUILD determines the row, column, length, and name of the userfield directly from the program screen. Other userfield attributes that you may change include:

1. TYPE - describes what kind of data goes in this field. The CHAR default can be changed to NUM for only numeric data, FIXED for only integers, FMT for a format specification, INPUT for the name of a SAS data set, VARLIST for one or more variable names, VARLISTC for character variables, VARLISTN for a numeric variable list, or NAME for a SAS variable name.

2. CAPS - causes the value entered by the user to be uppercased.

3. JUST - justifies text: L for left, R for right, C for centered.

4. PAD - is the default fill character used to initialize the field.

5. PROTECT - prevents the user from entering data in the field.

6. NON-DISPLAY - keeps entered data from being displayed. You might want to specify this attribute for a password field.

7. FORMAT - is a SAS format used to convert the value for display.

8. INFORMAT - is a SAS informat used to read the userfield value.

9. REQUIRED - causes an error message if the user does not enter a non-blank value in the field before leaving the screen.

10. LIST - lists the values used to validate the field. If the list of valid values does not fit on the line, then you may put the valid values in a LIST object. The LIST object reference is given by an equals sign in front of the list object name:

   = COLOR.LIST

The LIST attribute field may also be used to give a range of valid values. For example, if a numeric userfield should accept values between zero and one hundred, this list specification will do it:

   < 0 100

Note: if the userfield type is VARLIST, VARLISTC, or VARLISTN, then the list specification is used to tell the system on what data set(s) to look up each variable. Three forms are available:

a. an asterisk followed by the actual lookup data set name. * WORK.A means that the variables must exist on WORK.A.

b. an at sign followed by the names of INPUT userfields. The variables must exist on one or more of the SAS data sets referenced by the corresponding INPUT userfields. @ ds1 ds2 means that if the user keys "A" in INPUT field ds1 and
"B" in INPUT field ds2, then the variables must be on data set A or data set B to be valid.

c. if the list does not start with * or @, it is assumed that names of the type INPUT userfields are given. Each variable must exist on ALL data sets referenced by the INPUT userfields whose names you key on the LIST line.

11. HELP - is the HELP or MENU object name to be displayed when the user puts the cursor in the field and enters the HELP command. For example, suppose the user enters "APPLE" in the COLOR field and gets an error message. If the user then puts the cursor in the userfield and presses the HELP key, then you could display the list of valid colors in COLOR. HELP.

12. SAS MACRO VARIABLE - is the name of a SAS macro variable. When specified, PROC DISPLAY looks for the SAS macro variable, and if the SAS macro variable exists, then its value is loaded into the userfield prior to screen display. Any change to the userfield value will update the SAS macro variable's value also. This feature is useful for dynamic initialization of fields on the screen and for passing user-keyed information across procedure boundaries. For example, if you associate the COLOR userfield with a SAS macro variable called CLR before the program screen is displayed by executing the SAS statement

```sas
%LET CLR = BLUE;
```

then "BLUE" will appear in the userfield when the screen is displayed to the user. If the user changes the value of the userfield to "RED," then a subsequent reference to SYMGET('CLR') will return the value "RED."

There is a special userfield type, called ACTION, given to a one-character userfield consisting of a single ampersand. This type is given a default userfield name of FIELDn, where n is the userfield sequence number. Any non-blank entry in an ACTION field is converted to an uppercase X. An action field is useful where you want the user to select an alternative.

So far we have seen how to set up userfields with attributes for customized validation and processing of the data entered in them. Now let's look at another very powerful feature of SAS/AF program screens: conditional statement execution. You can cause SAS/AF software to ignore parts of the SAS code below the dashed line on a Program screen depending on whether or not a specified userfield value is blank. One level of conditional push nesting is possible. The verb "to push" here means "to send some SAS code to the SAS word scanner for execution." The five conditional push indicators and their meanings are as follows:

- `##ufname` push what follows if the value of `ufname` is not blank.
- `##ufname` push what follows if the value of `ufname` is blank.
- `ufname` push what follows if the value of `ufname` is not blank and a prior `##ufname` or `##ufname` has not suppressed pushing.
- `ufname` push what follows if the value of `ufname` is blank and a prior `##ufname` or `##ufname` has not suppressed pushing.
- `#` push what follows unconditionally.

We can modify our previous example to change the second PUT statement if the user does not key in a color:

```sas
DATA _NULL;
PUT 'Hello.';
@COLOR PUT 'Your favorite color is &COLOR';
#COLOR PUT 'You have no favorite color.';
# RUN;
```

In this example, either the second or the third PUT statement will be submitted to the SAS word scanner. Note that there is an assumed `#` in front of the SAS source below the line (the first two statements will be pushed unconditionally), and the `#` in front of the RUN statement is necessary to negate the effect of the `##COLOR` conditional push indicator.

Here is another example showing how nested conditional push indicators may be used. Suppose you have an application that prints a report and the user can specify what variables to include. The program object might look like this:

```
Special analysis for pilot study
You are about to run the statistical analysis for the experiment.
The summary file will be updated.
Do you want to include any particular variables in the report?
If no variables are specified, all variables will be printed.
Other variables: leave

Special analysis for pilot study
You are about to run the statistical analysis for the experiment.
The summary file will be updated.
Do you want to include any particular variables in the report?
If no variables are specified, all variables will be printed.
Other variables: leave

data pilot.reports;
keep pilot.mann pilot.trial;
by trials;
print proc print;
where not equals ;
## run;
```
In the example above, the DATA step will always be submitted when the user enters the END command on this screen in PROC DISPLAY, but if the action field FIELD1 is left blank, then both the PROC PRINT statement and the VAR statement will be omitted. If the action field FIELD1 is not blank, meaning that the user does want the report printed, then the PROC PRINT statement will be submitted. The VAR statement will be submitted if and only if both FIELD1 and PVARS are non-blank. The purpose of the nested conditional push indicators here is to avoid submitting a VAR statement without a PROC PRINT statement. Let me point out here that you would want to specify "¥ pilot. master" on the LIST attribute line for the userfield PVARS in order to make sure that all of the variables requested by the user exist on the input data set for the PRINT procedure.

I would also like to mention here that if the user cannot remember what variables may be selected, then he can enter the DIR command to see a current listing of the variables on the specified data set. And if the user selects this program screen on a menu and then changes his mind and decides not to run the analysis, then the CANCEL command will allow him to return to the menu without submitting any SAS statements.

There are two other ways for the user to leave a program screen without submitting any SAS code: if all of the SAS statements below the dashed line are within conditional push indicators and the associated userfield values keyed by the user cause all of the code to be suppressed, then the user will simply be returned to the SAS System. To avoid having the user leave your full-screen menuing system by ending a program screen without submitting any SAS statements, you can make the final part of the SAS code below the line a PROC DISPLAY statement to put the user back on your main menu:

```
PROC DISPLAY CATALOG=LIB.MEM.MAIN.MENU; RUN;
```

Then there's the "fast branch" facility on a SAS/AF program screen, where you want to give the user the option of going directly to another screen, bypassing the submission of SAS statements. To set up a fast branch, you put two greater than symbols ("=>") in columns 1-2 of the first line following the dashed line, immediately followed by a userfield name, followed by a space, followed by the name and type of the screen to branch to when the userfield value is not blank. For example, if you have a userfield called LEAVE where a non-blank value means that the user wants to go to another menu. Rather than submitting the statements "PROC DISPLAY C=LIB.MEM.OTHER.MENU; RUN;" you can follow the dashed line with this fast branch line:

```
=>LEAVE OTHER.MENU
```

and the user goes to OTHER.MENU without getting out of and back into PROC DISPLAY. You can have multiple fast branch lines, and the branch will occur for the first non-blank userfield specified. If all of the fast branch userfield values are blank, then we fall through into the SAS statements for possible submission.

Here is an example of fast branches on action fields to make a program screen act like a menu:

```
User, where do you want to go?
4 Home  5 Tips for the food
5 Enter del Fuego
6 Return to the operating system

=>FIELD1 PUMA PROGRAM
=>FIELD2 RUN.179
ENDSAS
```

Now, what if the menu items on your menu are not static? You can make the description lines into protected userfields and load values into them dynamically by associating the description userfields with SAS macro variables. You do not need to know all about the SAS macro language to use this feature. Just code a "%LET MAC=value;" statement or a "CALL SYMPUT(‘MAC’,’value’);" statement in a DATA step prior to displaying the program screen. For example, when a user selects "Vegetable" on a menu, you transfer to this program screen:

```
%LET VEG1=
%LET VEG2=
%LET VEG3=
DATA JULI-;
SET DAILY.VEGGUS;
IF JUG_1 THEN CALL SYMPUT(‘VEG1’,VEGETABLE1);
IF . . . THEN CALL SYMPUT(‘VEG2’,VEGETABLE2);
IF . . . THEN CALL SYMPUT(‘VEG3’,VEGETABLE3);
RUN;
PROC DISPLAY C=LIB.MEM.OTHER.PROGRAM;
RUN;
```

The program screen above has no userfields, so the SAS statements below the line are executed immediately. They access a file containing the names of today's vegetables in a variable called VEGETABLE, and each vegetable's name's value is put in a SAS macro variable. Then the user sees
VSELECT. PROGRAM, which looks like this to the user:

Please select a vegetable:
- Carrots
- Beans
- Cucum

The program screen VSELECT. PROGRAM looks like this to the builder:

Please select a vegetable:
1. Carrots
2. Beans
3. Cucum

The final step in Program screen development is to test it. To make sure that your program screen generates correct SAS statements, you can run PROC DISPLAY with the LIST option on in order to get the generated SAS statements on the SAS log. For example, if we have saved the new program object REPORT. PROGRAM and we want to see what SAS statements are generated when the user requests the report, we can enter

PROC DISPLAY LIST CATALOG=LIB.MEM.REPORT.PROGRAM; RUN;

and when the END command is entered, you can browse the SAS log to see the generated SAS statements from the program screen.

When the Program screen works OK, go back to PROC BUILD, edit the tested program screen and enter the command "DMS OFF" to tell the SAS Display Manager not to display any SAS log lines generated by the execution of SAS statements from a program screen. With DMS OFF, the user never leaves your full-screen menu-driven application, and he may not even realize that he is running under the SAS System. By putting your initial PROC DISPLAY statement (the one that fires up your customized application) in a source file that you copy in and submit using the AUTOEXEC facility of the SAS display manager, then the end user can start up the system from the operating system command line with a single command, and never see any SAS source code.

You can set up your LIST object for validating userfields by going to the catalog screen command line and entering "EDIT A.LIST" where A is the name of the list you want to create. Then you will see a system screen where PROC BUILD asks you about your list items' length, type, format, and informat. Then you get a data entry screen for the list.

The last screen type we will cover today is called CBT, for computer-based training. Even if you do not plan to set up in-house training courses for your users, there are some features unique to SAS/AF CBT objects that you may find useful for other applications. You will see those features first and then you'll see how easy it is to write your own courseware, customized to your shop's requirements.

A CBT object is a set of frames. A frame is a panel of text, like a Help screen, or a multiple-choice question. When PROC DISPLAY processes a CBT object, the end user can go from one frame to the next simply by pressing ENTER, but the system will stop on a question frame and wait for the user to answer it. So if you have an on-line presentation to give, you can put it into a series of CBT frames and display them to an audience one at a time simply by pressing ENTER.

When the last frame of a CBT object is displayed and the user presses ENTER, PROC DISPLAY takes the user to the CBT object's first child. You can specify a CBT object's child using the ATTR command in PROC BUILD. Typically a user will select a CBT course from a menu, and you want the student to return to that menu when the course is ended. But you can specify some other screen as the CBT object's first child and chain several CBT modules together so that the user goes from one CBT object to another automatically just by pressing ENTER.

Another unique feature of CBT objects you may find useful is the option to transfer from the CBT object to another object when the user selects an answer to a multiple-choice question. This feature might be used by a course builder to branch the student to a remedial lesson when the answer chosen indicates a complete lack of
knowledge of the course material. You can optionally specify that a checkpoint be taken when the branch out of the CBT object occurs; after the remedial CBT lesson is finished, you can branch the student back into the main lesson at the point where the student left off. By setting up a transfer object for every possible answer to a question, you can make a CBT object act just like a menu, except the user selects an item by putting any non-blank character in the action field corresponding to an answer.

Now let's see what is involved in actually building a CBT object. You enter "EDIT QUIZ.CBT" on the catalog display command line to bring up a blank slate for entering the lines making up each CBT frame. In fact, PROC BUILD treats a CBT object like a HELP object except that the ATTR command gives you a subpanel for entering the first child transfer object name, which is where the student goes when the course finishes displaying the last frame.

By the way, if your CBT screen name contains the letters QUIZ, then the student's score is written to the SAS Log when the course is ended. This is the only kind of course tracking available in SAS/AF software.

CBT frames that are not questions, but only displayed text, are separated by dashed lines. Each time the student presses ENTER, a page of frame lines is scrolled forward. By the word "page" we mean the number of lines that will fit on the user's terminal at one time. In other words, a CBT frame may have more lines than will fit on the user's terminal display, so the procedure breaks the frame into pages and displays the next page each time the user presses ENTER until the end of the frame is reached. The end of the frame is recognized by PROC DISPLAY when it sees a dashed line or a question mark in column 1 of a line. You may find it more convenient to make all of your frames a single page that can be displayed all at once to the user and to separate the frames with dashed lines.

There are two ways you can alter the way a CBT text frame is shown to the user. One way is by putting three not signs (.....) in columns 1-3 of a line, signifying that the lines from the start of the frame up to the three not signs should be treated like a title and always displayed. The three not signs say to the system, "No no no, do not scroll these lines."

Or you can put three at signs ((000)) in columns 1-3 of a line. Three at signs say, "At at, stop at this line." When the student presses ENTER, only the lines up to the @S line are displayed. The next time ENTER is pressed, the same lines are redisplayed with whatever lines you have following the @S line, up to and including another @S line, if any. This facility is used for showing a lot of material to the user on one frame, but a little bit at a time.

To start a question frame, enter a question mark in column 1 of a line, and put a digit in column 2 specifying how many tries to give the student to answer the question correctly. Then on the next line, key in the question itself, just the way you want it to be shown to the student from one to eight answers to select. Each answer is a single underscore as the first non-blank character on a line, followed by descriptive text. Following the answer fields, you specify what you want to do when the user chooses the answer. You have three actions that can be done when an answer is selected: transfer to another object and checkpoint the next frame for possible restart; transfer to another object without the checkpoint; or stay in the CBT object and give the user some feedback text.

When PROC DISPLAY displays the CBT object question frame, it looks for a response specification for each possible answer. Each response specification is started by a pound sign in column 1, followed by the integer representing the answer sequence number, followed by a capital "C" in column 3 for the correct answer. The same line, starting in column 5 or beyond, put "object.type" to transfer to that object with checkpointing, or "-object.type" to transfer without checkpointing. An alternative, displaying feedback text, is done by following the line containing the pound sign in column 1 with the lines of text you want the student to see after that answer is chosen.

The way the feedback text is displayed is different for the correct answer. For incorrect answers, the feedback text is displayed with a prompt for the student either to try again or to go on to the next frame if the number of tries has been used up. For the correct answer, the first line of the feedback text following the pound sign line is displayed only if the student selects the correct answer in the allowed number of tries. After the number of tries has been used up, or when the student backs up in the course to a previously answered question, only the second and subsequent lines of feedback text for the correct answer are displayed. That first feedback text line for the correct answer is good for some positive reinforcement for the student. Other feedback text lines are good for explaining to the student why the corresponding answer is or is not correct. Here is a sample question frame with four possible answers.
In SAS/AF Software, what do the letters AF stand for?

- Applications Facility
- Apple Farm
- Automatic feedback
- None of the above

No, abbreviations may not be registered for a trademark.
SAS/AF is a registered trademark of SAS Institute Inc.

Answer 1 gives the student some feedback for an incorrect answer.
Answer 2 branches the student to RETAKE.HELP with no checkpoint, so that a return to the CBT course starts at the beginning.
Answer 3 branches the student to INTRO.CBT with the checkpoint, so that the student may resume the course starting at the next frame.
Answer 4 is correct, and the student gets some positive reinforcement in line one of the feedback, followed by explanation text in subsequent feedback text lines.

Another type of CBT frame you might find useful is the unconditional branch, where you specify that a transfer to another object is to be done when the user comes to that point in the lesson. Put the letters "PPP" in column 1-3 of the first text line in a frame, followed by the name of the transfer object, and the student will go to that object automatically when the frame is processed.

We put this feature in so that the course that teaches SAS programming can let the student compose SAS statements and test them and see the results of the execution of those statements by the SAS System itself, and this objective is achieved by an unconditional branch to a program screen in the middle of the CBT course, with a checkpoint so that the student can see the results on the SAS log and then return to the lesson at the point following the PPP frame.

So there is SAS/AF Software. You have seen how easy it is to build different screen types and tie them together. Help screens can be used to display informative text to the user from any point in the system. A menu screen branches the user to other screens that you specify as menu children screens. Or the user can return to the parent screen with the END command. CBT screens are linked together in your system in a similar way with the parent and first child, plus, you have the additional flexibility of branching to other screens when a particular multiple-choice answer is selected or when a particular point in the lesson is reached. But the most powerful part of SAS/AF software is the program screen, where you can substitute validated userfield values directly into SAS program statements and, conditionally, push them for execution. All of the functionality of SAS/AF software for building your own customized full-screen applications is yours to enjoy.