INTRODUCTION

Today's industry professionals require a statistically powerful, yet easy-to-use data analysis and reporting tool that relieves them from the difficulties of software programming and syntax. Texas Instruments new Workstation Interface for Statistical Analysis and Reporting of Data (WISARD) software does just that by using SAS/AF to provide a high level interface to the many capabilities of the SAS system. WISARD guides the customer through topical menus and selections, then takes those inputs and logically navigates the intricate decision matrices and mutually exclusive conditions inherent in the SAS programming environment in a way transparent to the customer. As WISARD displays the tabular and graphical results of a selected function, it simultaneously provides in a preservable, standard template form the SAS program generated to achieve those results. WISARD imbeds clear documentation throughout the generated program, enabling the customer to easily modify and/or re-use the program at any time. To illustrate the high quality of interface achievable using SAS/AF software, this paper describes the conventions and tools WISARD uses, along with instructions and advice on their creation, then concludes with an investigation of several of the implementation issues encountered.

I. CODING METHODOLOGY

The purpose of this section is to give practical instruction, advice, and a few tricks in developing many of the same useful features that WISARD incorporates. The acronym SCL will be used as a reference to Screen Control Language.

A. SHORTCUTS

Utilizing the following four SAS/AF entities will save much wasted time in producing the SCL code to duplicate their functionality.

ATTR screens deal with specific fields. ATTR screens may be used to rename a field for SCL usage, capitalize, protect, justify, require, color, error-check, and give initial values for a field, among other things.

GATTR screens allow designation of screen titles, screen size, parent screen in the screen hierarchy, and a help screen for the current function, plus more.

LIST entries may be used to ensure that a field's value is found in a list of possible values for that field. An automatic error message can be generated for exceptions.

KEYS entries allow selected function keys to be specified for each individual function. Function keys may serve as good substitutes for line commands or screen fields. Customized non-SAS function keys may also be created and will be discussed later.

B. INTEGRATING SAS/AF WITH SAS

In addition to SCL, SAS language itself may be used in SAS/AF applications. By using SUBMIT-ENDSUBMIT blocks with the CONTINUE option, an application can run SAS in the 'background', invisible to the SAS/AF customer. One application in WISARD uses this concept to run PROC MEANS to generate the minimum and maximum value for a pre-selected variable. The results of the PROC MEANS procedure are then placed into global variables at the end of the procedure during execution, read in by WISARD, then placed on the screen. Results of a SAS procedure are best read back into a SAS/AF application by means of global variables, or outputting to and reading in a dataset of the results.

Some products and procedures, such as SAS/FSP with menus or the GREPLAY full-screen procedure, are sufficient stand-alone products. By submitting their 'startup' syntax from within a SUBMIT-ENDSUBMIT block, products such as these can be transparently accessed by a SAS/AF application. However, the customer must use the 'END' command to return to SAS/AF, as simply entering the 'AP command will not kill such procedures as full screen GREPLAY.

A very ergonomic tip is to transport the customer to Display Manager by use of a 'DM' statement whenever he executes an application. This way, the customer can observe the generated code being processed in the LOG window as SAS is interpreting the code. This avoids forcing the customer to stare at a seemingly 'dead' screen while waiting for SAS to complete execution. The customer may easily return to the SAS/AF application by using the 'AF' command.

C. CREATING YOUR CONVENTIONS

TOGGLE SWITCHES: Using REVERSE video, type in a variable field. Create an adjacent protected field in non-reverse video for the value to be toggled. When the program executes, place a key word in the reverse video field and an initial value in the non-reverse field.
Use the CURWORD function to determine if the customer has hit the ENTER key with the cursor on the key word in the reverse video field. If so, use an 'if-else' statement to swap the value in the non-reverse field.

BUTTONS: Created exactly like toggle switches (i.e. with a variable value written in reverse video), except that when the CURWORD function detects that the cursor is on the key word placed in the variable field by the program, the resulting action taken is up to the discretion of the programmer.

CUSTOMIZED SELECTION LISTS: This trick involves making a data step in base SAS. Type all selections in an INPUT statement in the data step. Then use the CARDS delimiter, but do not type in any cards; just immediately follow the CARDS delimiter with another semicolon. This makes a dataset having as variable names the selections, with no observations. Then, once in SAS/AF, use the OPEN function to open the dataset of selections. Use the VARLIST function to enable the customer to pick variables, which are really just the names of the selections. The VARLIST function returns a string containing the names only of selected variables (items) which can then be parsed out using the SCAN function.

POP-UP WINDOWS: To simulate a pop-up window, use a CALL DISPLAY statement to call a new 'pop-up' screen from the current screen. Use the GATTR window on the new screen to designate a smaller screen size for the new screen. Place any selections, results, etc. from the new screen into either global variables or datasets, then read them when control is returned to the calling screen. This is really just one program calling another, but with screen sizing and transporting information back and forth via global variables and datasets, the illusion of a true pop-up window is effectively achieved.

CUSTOMIZED FUNCTION KEYS: Use a KEYS type SAS/AF entry to enter in the function keys. Use the VERIFY OFF option in the KEYS window if any new keys are not standard SAS syntax. Upon executing the SAS/AF application, pressing a function key generates that key's contents at the command line, although 'invisibly'. Use the WORD function in SCL to retrieve the command line entities and then subsequently act on them.

CUSTOMIZED LINE COMMANDS: Use the WORD function to pull the contents of a line command into SCL when the ENTER key is pressed. Do a string comparison on the results of the WORD function to determine if the command entered is the customized command. If so, act accordingly.

SCROLLING SCREENS: For multi-screen applications, the top part of a screen may remain fixed while the bottom portion of the screen may be scrolled. Place three carats (^^^) at the leftmost side of the screen on the line directly below the portion to be fixed. Designate each succeeding scrollable panel by a row of dashes all the way across the screen between each panel. Each panel must have either less than or equal to the amount of lines below the carats to the bottom of the screen on the top screen. During execution, use the FORWARD and BACKWARD commands or Page-Up and Page-Down keys to scroll the panels.

EXTENDED TABLES: For pulling in information from datasets or SAS procedures with varying lengths or results, use an extended table to fluctuate with the amount of the incoming data. See the section on implementation issues for a method of handling extended tables. An extended table is another tool already provided by SAS/AF.

ENTRY FIELDS: Entry fields may be checked via the LIST and ATTR entries, or verified by SCL statements. Manual entry fields are useful when a customer already knows what he wants to select from a large list and would rather quickly enter it himself instead of scrolling the list.

DISAPPEARING FIELDS AND BUTTONS: Fields can be made to 'appear' and 'disappear' on the screen by initially giving the field the same color as the background color. Designate a different color on the ATTR screen for that field. To make the field 'appear', use the ERRORON function for the field. ERRORON causes the field to become the color and attribute of its error designation on the ATTR screen. The ERROROFF function returns the field to normal. The PROTECT and UNPROTECT functions are useful for protecting an 'invisible' field and unprotecting it when made to 'appear'.

IDENTICAL BUTTONS ON ONE SCREEN: This trick involves creating variable fields that have only a portion of the field in reverse video and the other portion colored the same as the background color of the screen. During execution, the reverse-video portion of these fields may have the same text in them, but the 'hidden' portion of the fields must contain the differences. Use the CURWORD function on the entire value of the field. For example: In two five-character fields, the first four characters are reverse video. Put the text 'EDIT1' and 'EDIT2' in each field, respectively. The customer will only see the 'EDIT' portion, as the numbers 1 and 2 are the same color as the background. Then use the CURWORD function to check for 'EDIT1' and 'EDIT2', not 'EDIT'.

TOGGING THE FUNCTION KEY DISPLAY: Do not type the function keys directly on the screen; put a variable field there instead. Inside the program, fill in the variable field with a string listing the function keys.
Create a unique line command set, such as KEYSON and KEYSOFF. When the KEYSOFF line command or function key command is detected by using the WORD function, simply blank out the variable field holding the keys display, and vice versa for the KEYSON command. You may set a global variable switch whenever either command is entered. New programs can then read this variable upon initialization to determine whether to start the screen with the keys field filled in or blanked.

D. SCREEN-JUMPING

Screen-jumping denotes the ability to enter a screen identification name at the command line of the current screen and 'jump' (transfer) directly to that screen. The first model of screen-jumping presented will be simple. The second, which WISARD uses, is more complex.

For simple screen-jumping, give each program entry that the customer should be able to jump to a one-word screen identification name (code name). Build a macro with if-then-else logic to determine if a code name has been entered on the command line (use the WORD function). A pseudocode version might read "If (return of the WORD function) equals (code name) then CALL GOTO (the code name screen) else (check next code name, etc.)." Do this for each code name in the list. On the GATTR screen for each program entry, use the space provided to list a parent entry for that screen. This is necessary because CALL GOTO breaks all screen hierarchy and erases parentage. Lastly, place the screen-jumping macro in the MAIN section of each program that should have screen-jumping capability.

For complex screen-jumping, divide all code names into categories such as APPLICATIONS and UTILITIES (WISARD uses these). In the screen-jumping macro, if an APPLICATION's code name is entered, use a CALL GOTO to jump to it. If a UTILITY's code name is entered, use CALL DISPLAY. With CALL DISPLAY, control will return to the calling screen after exiting the called screen. CALL DISPLAY also prevents the calling screen from re-initializing; i.e. the condition of the calling screen before making the screen jump will be preserved upon return. The basic concept is that an APPLICATION should be able to call any number of UTILITIES while working with the APPLICATION without re-initializing that APPLICATION upon return from a UTILITY.

UTILITIES should not have parent screens designated on the GATTR window, as this will conflict with the CALL DISPLAY function. APPLICATIONS should have parent screens specified.

The code will need to contain logic to check for the following condition: If a jumped-to UTILITY calls its own parent (the calling program), which it would naturally return to anyway, then do not execute a CALL GOTO to return to the parent screen, but simply END the current screen instead.

The most difficult logic for complex screen-jumping is this: When jumping from one UTILITY to another UTILITY, do not directly CALL DISPLAY the second UTILITY. END the first UTILITY, return to the calling screen, then jump to the second UTILITY from the calling screen. Global variables are used to pass the screen code name and destination of the second UTILITY back to the calling screen. This method of returning to the calling screen for each new UTILITY jump holds the stream of active screens to two; otherwise, calling several UTILITIES in a row would result in having to backtrack through each of them in order to return to the initial calling screen. Also, this greatly conserves memory by holding a maximum of only two screens/programs in memory at once instead of holding a large stream of many consecutively called screens/programs.

E. USING ARRAYS WITH SAS/AF

Arrays may be used effectively when dealing with screen fields. Create a set of similar fields with the same name plus an index number, such as NAME1, NAME2, NAME3, etc. An array called NAME in SCL can then be used to control these screen fields. For example:

```bash
do i = 1 to dim(NAME);
    NAME(i) = ":" / blanks all 3 screen fields /
end;
```

SCL arrays are not directly interpreted in SAS submit blocks. For instance, in the following statement:

```bash
submit;
&NAME(1);
endsubmit;
```

NAME(1) will not be interpreted as a SCL array item. An intermediate step will resolve the interpretation:

```bash
buffer = NAME(1); /* place in Intermediate buffer */
submit;
&buffer; /*use buffer instead */
endsubmit;
```

II. SPECIFIC IMPLEMENTATION ISSUES

The following sections deal with several of the more prominent issues encountered in the development and implementation of WISARD.

1. Once a customer moves to Display Manager during execution of a SAS/AF application, he is not under the control of the application. This
may result in the customer tampering with pieces of data or datasets that the application would need. Take care to check for these conditions by monitoring file statuses and data values immediately before and after leaving a SAS/AF application for Display Manager interaction.

2. Communication with Display Manager is difficult in some areas. Retrieving the contents of the LOG window or output from the OUTPUT window are two major areas of difficulty to which there are no easy alternatives. Placing the output from a procedure into a dataset and reading from that dataset seems to be the best way to retrieve SAS results, although a bit indirect in concept.

3. Some timing differences may exist between SAS/AF applications and SAS procedure/Display Manager interaction. For instance, a SAS/AF application that performs some primary action, then executes a SAS procedure or Display Manager action sequence before returning to the SAS/AF application for a conclusive action, such as displaying a post-processing message, may proceed out of coding sequence by displaying the post-processing message first. Use of the REFRESH command directly after the non-SASI AF portion of the processing may alleviate this condition at times.

4. SAS/AF programs under version 6.03 for the PC have an object code limit of 32756 kilobytes.

5. Dynamic array allocation is not possible. All array sizes must be predetermined prior to initial program compilation.

6. Extended tables and multiple screen applications with fixed window portions may not coexist, as the demarcation symbol (***) for an extended table area and the end of a non-scrollable window area is the same and may be used only once.

7. Extended tables do not process in sequential order. The information is not written to the table until the MAIN label is completely through executing. Therefore, values supposedly written to the table may not be retrieved or acted upon and then rewritten all in the same cycle. A suggested remedy is to immediately place the contents of the extended table in an array in the PUTROW section. Do all processing on the array instead of the table locations. Put the values back into the table from the array at the beginning of the GETROW section.

**CONCLUSION**

SAS/AF is an excellent package for developing customized applications. Desired application results can almost always be achieved with the tools, functions, and SCL code provided in the SAS/AF software, with a pinch of ingenuity added. The Texas Instruments WISARD software was discussed to demonstrate the sophisticated interface levels possible with the SAS/AF product. This paper also provided some general information on working with the SAS/AF package, indicated which tools are best suited to specific tasks, provided practical instruction and tips for creating unique and useful personalized features, and highlighted a few key implementation issues, providing solutions where appropriate.

Contact the author at:
Texas Instruments Incorporated
Post Office Box 869305, MS 8437
Plano, Texas 75086

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