INTRODUCTION

When a new system is brought into the office environment, the user, regardless of skill level, is required to learn a new "language". The overuse of jargon in the system screens may cause the user to resist the new product. Several other factors that may cause the user to resist the system are:

- difficulty in relating their actions to the results they have received from the system,
- difficulty with making fine distinctions where different actions seem to accomplish the same result (or different results may require the same action), or
- difficulty in switching from the mere human "learning by trial-and-error" strategy to the computer's "error prevention" strategy.

If the user finds the system complex to use, then the use of it becomes a punishment. The confusion, frustration and panic caused by using the system blocks the user's concentration and may result in higher error rates, poor performance and job dissatisfaction. Over time, the system may be only partially used (only those features that were easiest to use or most beneficial) or it may be completely abandoned or rejected by the user.

In an effort to avoid this system "death", this workshop, using SAS version 6.04, will highlight several of the techniques discussed in this paper.

SCREEN LAYOUTS

Time and thought is needed in designing a generic layout for the system's screens. The task is to identify and reserve specific areas of the screen for certain kinds of information --commands, error messages, titles and subheadings, data fields, use of icons, fields/areas where users type, fonts, colors, style. The programmer should first focus on the format and the content of the screen itself and the information it contains, rather than on the code to generate it.

Generally, the screen should have an orderly, clean and clutter-free appearance. There should be an obvious indication of what is being shown and what should be done with it. Once into the system, there should be a simple way of making the desired selection and a simple way to get out the information needed. There should be a clear indication of when an action makes a permanent change in the data. And finally, an easy way should be provided to exit from the screen or the system if the user gets into trouble.

The programmer's goal is to guide the user through the screen (either horizontally or vertically). The user should be able to at a glance see where they are in the system or what the system is doing beyond their control. Several other considerations for good screen design are:

DO NOT

- use jargon, words or terms that are unique to the computer profession -- use the metaphor dictionary (see Fastullo & Garrido on how to build a metaphor dictionary),
- make up words to describe special functions or conditions,
- hyphenate words between lines.

DO

- use standard alphabetic characters to form short and familiar words (avoid contractions, short forms, suffixes and preixes),
- use simple, action sentences (do not forget the period at the end of each sentence),
- include no more than 40-60 characters on each line (or double column of 30-35 characters separated by five spaces),
- place information that must be remembered at the end of the text,
- use upper case letters for titles and labels,
- use white space on the screen to strive for balance, regularity and symmetry, and
- avoid humor and punishment.

In addition, make use of contrasting display features to call attention to different components on the screen, items being operated upon or urgent items. Some techniques for attracting attention are using high brightness, reverse video (useful with error fields or identifying the information being acted on), blinking cursor (remember to turn it off once the user has responded), color and audio (soft tones for regular, positive feedback, harsh tones for warning and error conditions). However, make conservative use of these features. Too much can disturb the user by reducing legibility and increasing eye fatigue or may embarrass the user by having the machine beep for every single error or response.

The above general techniques are often recommended for the development of any user interactive systems. The following sections deal with the implementation of user-friendly SAS techniques.

THREE TYPES OF ALERTS

There are three types of alerts: dead-end alerts, confirm/cancel alerts, and passive/progress alerts. If we rank the alerts with dead-ends being the most important, from the system's point of view, to passive/progress being the least important, then the design, colors and placement of the windows on the screen can be
done appropriately. For example: dead-end alerts belong in the middle of the screen in red, a more alarming color, and the passive alerts at the bottom in blue. In order to place the alert window on the screen in a strategic place, use the WREGION Screen Control Language (SCL) function. Suppose your screen device has 24 lines and 80 columns, to position the alert, which consist of 2 lines of text, in the middle of the screen use the following SCL statement before the call to the alert program.

```
CALL WREGION(11,1,4,80,');
CALL DISPLAY('LIBNAME.CATALOG.ALERT.PROGRAM',
    message,color);
```

This will place the window starting on line 11 column 1, with 2 lines and 80 columns. It should be noted that 2 extra lines are required for the top and bottom borders of the window. The code above assumes the alert program is "generic" code that will accept parameters to customize the window with message and color. Dead-end alerts are windows that are displayed when user intervention is required. These are usually serious errors and the system can not continue without additional information. For instance if the user requests a print by the variable race and that variable did not exist on the data set, the window could point out the error and ask for an alternate by-variable. These types of alerts are best handled by calls to display additional SCL created windows which will not leave the screen unless the user satisfies the system's request for information.

Confirm/cancel alerts are windows that are displayed when a users request is unusual or questionable, or at a decision point. For instance if a user requests a print of a very large data set, the system may wish to ask the user to confirm the print. This would also be similar to many systems that request confirmation for the deletion of files. Before a user makes a permanent change to a data set, notify him/her with an appropriate message in the confirm/cancel window and ask for confirmation before proceeding.

Passive/progress alerts are not necessary for the system, but are quite helpful to the user. If a procedure takes a considerable amount of time, it is a good practice to let the user know the procedure is working, but time consuming.

**EASE OF DATA ENTRY**

Ease of data entry is another important quality of a good system. When your users are required to enter data, determine if there is a finite set of answers to a question. Depending on this number of possible answers, the developer may wish to provide the answers in the form of a list. Providing the answers in the appropriate format will reduce the number of data entry mistakes and make the system more user friendly. Lists are useful when the number of responses is large or is expected to grow, like customer address list.

There are several ways that a developer can help the user determine valid responses in an application by only using the general attribute and attribute windows. In either window, the developer can define an entry which contains specific or general information, to be displayed. This can be accomplished by placing the name of the help screen in the help field.

Help: **MAIN** **HELP**

In the above example, MAIN.HELP could contain a list of valid responses.

Another helpful task is to have an initial value in a field when the panel is first displayed. The developer can easily do this by declaring a value in the initial attribute field in the ATTR window. In execution mode, the value specified in this field will appear on the panel.

On the other hand, by making use of the screen control language, there are several statements and functions that the developer can use to check the values on the screen. Of particular usefulness is the CURWORD function which returns the string or word the cursor is currently resting on — the system can "read" this value without the user needing to key in any fields:

```
CHOICE = CURWORD();
CALL SYMPUT('value',CHOICE);
```

As another technique, the developer may decide to create application specific commands. The WORD function in SCL captures the word(s) on the command line. By adding two lines of text to the bottom of the display screen to read:

```
Enter DISK on the command line to print to disk.
Enter PAPER on the command line to print to paper.
```

the developer can create two new commands. The following SCL code could be used to execute those commands. This code would be especially handy stored as a macro, since it should be placed in both the INIT and MAIN sections of the SCL program.

```
IF UPCASE(WORD(1)) = 'DISK' THEN DO;
    CALL NEXTCMD();
    CALL ... (SCL code to perform command) ;
    END;
ELSE IF UPCASE(WORD(1)) = 'PAPER' THEN DO;
    CALL NEXTCMD();
    CALL ... (SCL code to perform command) ;
    END;
```

**WORKSHOP GOALS**

This workshop will present techniques for making a system user friendly. The attendees will participate in inserting SAS code into an existing system.

The following ideas will be demonstrated:

A. In the general attribute panel:

- Create a customized display when the "help" command is entered.
- Identify a name for each display panel (this name will be displayed in the upper left corner).
- Specify where on the screen a panel will be displayed.

790
B. In the attributes panel:
   - Define a list of valid values for a field.
   - Set the initial value for a field when a panel is first displayed.

C. In the source panel:
   - Use screen control language to create a "point and click" screen where the user does not have to enter a value.

ACKNOWLEDGMENTS

The author wish to express thanks to the ARC Professional Services Group for their review of this paper. He also wishes to express special thanks to Kita Garrido and Veronica Fasullo for help in developing this paper.

The author may be contacted at:

ARC Professional Services Group
1301 Piccard Drive, 2nd Floor
Rockville, MD 20850
(301) 258-2688

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

REFERENCES


791