

INTEGRATING USER SUPPORT INTO SAS APPLICATIONS

Michelle Fruchtenicht and Eric Brinsfield

Meridian Software, Inc.

Introduction

SAS/AF and SAS/FSP software enable you to build and incorporate a variety of online user support features, such as help windows, embedded training, online documentation, and error message windows. Despite some inevitable tradeoffs (such as completeness of coverage versus ease of use), the user support you provide in your SAS applications is limited mostly by your imagination.

In this paper, we hope to stimulate thoughts about new possibilities for online user support by exploring some help features we've created at Meridian Software, Inc. for various applications.

We'll start by presenting the general principles we follow for designing user support. Then we'll suggest ways of using SAS/AF software to create help features that embody these principles.

You'll see, in general terms, how to build one version of each of the following features:

- selection lists
- help windows with links to related information
- field-specific help
- help indexes
- multiple levels of help
- online documents with text search capabilities.

General Design Principles

Designing good user support isn't a random process. To communicate effectively with users, you need to begin with clear design goals that focus on the user, not simply on the application itself.

At Meridian Software, we use the following six principles to guide our development of user support features:

1. Start with a good interface. Help systems and other user support features shouldn't have to make up for deficiencies in interfaces.

2. Determine who your user groups are and what types of support they need. Consider supporting the job, not just the application.
3. Provide specific support at the point of need, while leaving the user in control. That is, narrow the information the user has to process and navigate, but provide open access to other help information as well.
4. Make help universally available as well as easy to access, navigate, and apply.
5. Write help information that is accurate, complete, clear, and well-formatted.
6. Prototype, test, and revise.

As we build user support features in this paper, we'll examine ways in which these principles affect design decisions.

Targeting Types of Support

Once you determine user needs, you can begin designing the appropriate user support.

User support generally falls into the following categories, each progressively more removed from the original information need:

- "interface enhancement" – improving the interface itself to automatically provide known information the user needs
- context-sensitive help that the user requests
- related information that the user requests, which may cover areas outside the application proper (such as prerequisite skills or information).

A rule of thumb is to answer information needs as close to their source as possible, using the most direct means available. If an error message would solve a problem, don't make the user access a help window for the information. (For example, we've started using message

windows to display more information than the message line holds.)

Determining what support to provide is relatively easy when user needs fall into the interface enhancement category. If your application can determine what the user needs, you are responsible for providing it as part of the basic interface (general design principle 1). Likewise, whenever you are relatively sure that the scope of information needed is restricted to an area of the display (such as entering values in a specific field), you can generally provide context-sensitive help in a single window.

More problematic are circumstances where you can't pinpoint errors or determine context precisely. In such cases, you're forced to fall back on more general help, which addresses what you assume (rather than what you know) the user needs. Only by observing representative users working with your application can you really determine what information you provide and the form in which you implement it.

Generally, though, in the following situations you can successfully apply the user support features we'll build later in this paper:

Selection lists (lists of items the user can select, which are subsequently processed by a program): Provide these whenever your application knows which values are valid for a field. Selection lists are crucial in data entry applications.

You can create selection lists containing fixed sets of items, or you can use SCL (Screen Control Language) to produce data driven lists that are dynamically updated.

Help windows with links to related information: In most applications, you should provide additional information or successive levels of detail. Often one window won't suffice. At a minimum, users should be able to determine the purpose and scope of the application, the mechanics of using the application, and the additional information available.

Field-specific help: At a minimum, offer this wherever users have to enter values without a selection list.

Field-specific help is especially useful if a field is related to other fields, or if users must enter values in a particular format or unit of measurement.

Help indexes: Why not always provide an index? As mentioned above, you can't entirely predict what information users want, but you can enable them to find it.

With a little more programming, you can provide an index that permits the user to add new subjects or search strings that they think are necessary.

Multiple levels of help: Consider offering different help levels when you have diverse user groups or when a user's needs will change over time.

You can create levels of help tailored to user needs along any of several lines, including

- expertise in using the application
- the user's purpose or task (such as data entry versus project management);
- the degree of detail desired.

Online documentation with text search capabilities:

When users need descriptive information (as opposed to directions for performing tasks) not easily predicted by context, consider an online document.

Online documentation is especially useful for users who want to locate instances of the same text throughout a document. Traditional indexes in printed documentation can't easily provide this feature at the same level of detail. For example, users may want to locate all occurrences of data set or variable names, or headings such as Caution or Warning.

Creating the Help Features

Generally, SAS/AF and SAS/FSP software provide several ways to create and implement any given user support feature you choose. In this paper, we show you only one way to build each type of user support feature. At the end of each example, when applicable, we list alternate approaches and indicate when you might want to use these methods.

Space considerations also limit our discussion to the design features critical to the user support feature at hand.

The explanations in this section assume that you know how to build catalog entries and that you are familiar with using SCL with SAS/AF and SAS/FSP software.

If you need to brush up on these topics, see these publications:

- *SAS/AF Software: Usage and Reference, Version 6, First Edition*, chapters 3, 5, and 7 (for building PROGRAM, CBT, and HELP entries)
- *SAS/FSP Software: Usage and Reference, Version 6, First Edition*, chapters 8 and 10 (for creating customized FSEDIT displays and using SCL)
- *SAS Screen Control Language: Usage, Version 6, First Edition* (for using SCL with SAS/AF and SAS/FSP software).

Selection lists

We frequently use selection lists, particularly to provide valid variable values for users editing SAS data sets displayed in customized FSEDIT windows.

An easy way to create a selection list is to use the DATALISTC function (for character values) or the DATALISTN function (for numeric values). These functions automatically generate windows containing the selection lists. You can supply a one-line custom message to users at the top of the window.

In the following example, users type a question mark (?) in the field to display the selection list. Once the user selects a value, the selection list window closes and the value appears automatically in the field.

To build the selection list, follow these steps:

1. If you're calling the selection list from a PROGRAM entry, select the NOPROMPT option in the ATTR window for the field where you're hooking up the help. (If you're calling the selection list from a SAS/FSP SCREEN entry, you don't have to do anything in this step.)
2. In the calling PROGRAM or SCREEN entry, add the following SCL statements to the MAIN section (this example calls a selection list for the character field MODELNUM):

```
if index(modelnum, '?') then do;
    modelnum=datalistc(dsid, 'valmnum',
```

```
    'Select one model number:', 'Y');
if modelnum^=' ' then
    _msg_=' Model number'      ||
    modelnum                  ||
    ' selected.';
end;
```

Other possible approaches:

Generally, we prefer to create a SAS/AF PROGRAM entry defined as an extended table to display the selection list, rather than using SCL functions such as DATALISTC or DATALISTN. Using a PROGRAM entry rather than a system-generated window allows you to control all window features and keep them consistent with the remainder of your application.

General help with links to additional information

For our general help windows, we typically use SAS/AF CBT entries because they're simple to build and modify. They lack most of the programming capabilities of PROGRAM entries, but for purposes of general help (without specific error conditions or contextual requirements), they work well.

CBT entries provide several types of branching useful for help systems, and they allow you to package separate frames of help together in a single entry. You can then access each frame independently and control whether users can scroll between frames.

In the example below, the CBT entry contains "goback" branches which act as links to other information. This information can be frames within the same entry, frames within other CBT entries, or a different type of SAS/AF entry altogether.

To access other information, users simply select text on the screen (determined by row and column coordinates you supply). When users close a window displayed by selecting one of these links, they return to the original CBT entry containing the link.

Each window accessed by a link can in turn contain other links. In effect, you can build as many connections between items of information as you want (being careful not to provide so many links that users get lost).

To build a CBT entry with goback branches to other information, follow the steps below. (For a complete

description, see *SAS/AF Software: Usage and Reference, Version 6, First Edition*, pages 121-125).

1. Build a CBT entry whose first line contains a question mark (?) in the first column and a 1 in the second column. The question mark serves as a frame indicator, and the 1 indicates the number of selections the user can make at a time.

Optionally, you can specify a name for the frame using the NAME= option on the frame indicator line. To prevent users from scrolling out of this frame, you can specify the LOCK option.

2. Type the general help text anywhere else on the screen. Then identify the row and column coordinates for any text you want to use as links. (The text must have at least one space before and after it, or the links won't display properly.)

Tip: If you position your cursor and issue the CURPOS command using a function key, the message line displays the cursor's absolute row and column position. However, you must determine the relative row number (NOT including the first row, which contains the frame indicator) in that frame only.

3. In the line immediately following the last row of text, specify a goback branch for each link in the following form:

```
#1 >< entry-name.entry-type
      frame=frame-name
      select= (begin-column,end-
              column,begin-row,end-row)
```

Example:

```
#1 >< moreinfo.cbt frame=overview
      select=(14,45,11,11)
```

(Note that the code for each goback branch must appear on a single line. Here we don't have room.)

In more detail, here's how you specify the goback branches:

On a single line for each link, type a pound sign (#), the sequential number of the branch, at least one space, and the goback branch indicator (><).

Then type the two- or four-level name of the target entry. If the link accesses a specific frame in a CBT entry (which must be named using the NAME= option), specify the frame name using the FRAME= option after the name of the target entry.

Finally, use the SELECT= option to specify the column and row coordinates for the text users select, in the form (begin-column,end-column,begin-row,end-row).

This all sounds complicated, but is in fact simple after your first try, especially if you repeat lines or cut and paste the goback branch specifications.

When you create this type of entry, consider possibilities for specifying links other than text in sentences. For example, you can create maps of your application or character graphics that users can select.

Other possible approaches: SAS/AF HELP entries (particularly for text searching or non-dynamic selection lists – see [Online documentation](#) below) or PROGRAM entries (to provide full programming capabilities).

Field-specific help

For PROGRAM entries with several input fields, you can save time and trouble by storing all help in a single CBT entry with multiple locked frames (see [General help](#) above). You can then access individual frames for field-specific help by specifying the frame name in the ATTR window for each field.

Storing all help in a single CBT entry makes editing and general maintenance much simpler, especially if you give the CBT entry the same entry name as the PROGRAM entry it supports. In addition, you can assign frames the same names as the fields they support. (A bonus: programmers find it easier to supply help information for windows if your naming conventions are this easy to follow.)

To display help, users position the cursor on the field and press a function key assigned the HELP command.

To create field-specific help as described above, follow these steps:

1. Create the PROGRAM entry that will call the help.

In the **Help** field of the GATTR window, type the name of the CBT entry.

In the **Help** field of the ATTR window for each field, specify the field's alias in the first of the two blanks, leaving the second blank empty. This is an example for the field DSNAME:

Help: DSNAME

2. Create a CBT entry as described in step 1 of **General help** above, using the NAME= and LOCK options for each frame. Specify the corresponding field alias as the value of the NAME= option.
3. Following the last line of text on the first frame, specify a goback branch corresponding to each frame (see step 3 in **General help** above). Because you aren't creating links in this case, omit the SELECT= option. Use the value of the NAME= option for each frame as the value of the FRAME= option. In addition, for each goback branch, add a MENU= option with the same value as the FRAME= option.

Example:

```
#1 <> window1.cbt
      frame=dsname
      menu=dsname
```

(The code for the goback branch must appear on a single line in the CBT entry.)

Other possible approaches: separate HELP, CBT, or PROGRAM entries specified in the ATTR window for each field (when fields require different functionality in help windows); SAS/FSP SCREEN entries (when you want to display a customized data set display as help information).

Help indexes

A help index solves many of the problems associated with providing general help – although it may take an extra keystroke or two, users can find exactly the information they want.

To create a help index, we use a PROGRAM entry specified as an extended table. The extended table

displays dynamically generated values (in this case, the help index items). You can call the extended table from any SAS/AF PROGRAM entry or SAS/FSP SCREEN entry in your application.

To provide the values for the table, we create a SAS data set, which we can update with new index items.

To create a help index, follow these steps:

1. Create a permanent SAS data set with the following variables:

Variable	Type	Length
INDXTERM	Char	less than 50
ENTRY	Char	8
FRAME	Char	8

The INDXTERM variable can be as long as the longest possible field on your display, but a very long field is visually overwhelming and forces the user to read a long line of text.

2. Determine the list of items to display in the help index. Then, using the DATA step, the FSEDIT procedure, or the FSVIEW procedure, add one observation per item (the value of INDXTERM) to the data set.

Identify where information about each item is located in your help information and enter the CBT entry name in the data set. For individual frames in CBT entries, specify the frame name as well.

3. Build a PROGRAM entry as an extended table with one field, INDXTERM, the same length as the INDXTERM variable. (For details, see *SAS Screen Control Language: Usage, Version 6, First Edition*, chapter 12.) To supply values for the table, open the data set you created in step 1 above.

To display entries when the user selects an item, fetch the selected row from the data set in the MAIN section of the SCL program. Then specify the following statement:

```
call cbt(entry,1,frame);
```

When users close an entry displayed by selecting an item from the help index, they return to the index.

Multiple levels of help

Most of our applications are used by different groups within an organization, such as data entry personnel, managers, and programming staff. To solve the problem of providing appropriate help information for various audiences, we give users a choice of help levels based on job category or experience.

On the main menu of the application (but in theory, anywhere), we provide two fields which act as toggles. At any time, users can select help that offers more detail (NOVICE) or less detail (EXPERT).

When users select a help level, the application sets a macro variable which points to one of two catalogs. Entries in both catalogs have identical names but contain different help information. When the user requests help, a master PROGRAM entry determines which catalog and entry to display.

To keep programming time short enough to make this feature feasible, we adopted a simple structure for the help system. For each program in our application (all of which are PROGRAM entries), we created a single like-named CBT entry.

To provide multiple levels of help, follow these steps:

1. Develop sets of help information (as many as you like) in separate catalogs. Important: All entries must be CBT entries with the same name as the PROGRAM entries they support.
2. Create a PROGRAM entry with a blank DISPLAY window (or an SCL entry), containing an INIT section similar to the following:

INIT:

```
control always;
helplev=symget('helplev');
caller =symget('caller');
callname=scan(caller,3,'. ');
helpname= 'catlib.' ||
          trim(helplev) ||
          '.' ||
          trim(callname) ||
          '.cbt';
call display(helpname);
call execcmd('end');
return;
```

(The MAIN and TERM sections can be empty except for the section labels and RETURN statements.)

3. In the GATTR window of each PROGRAM entry in your application, assign the name of the entry you created in step 2 as the value of the Help field. In the INIT section, add the following statement to store the current program name in macro variable CALLER:

```
call symput('caller',
            screenname());
```

Using the SCREENNAME function keeps the code generic in case you rename or clone an entry.

4. In any PROGRAM entries where you want to allow users to change help levels, create a field for each level offered. Specify the fields as members of a choice group. (For details on choice groups, see *SAS/AF Software: Usage and Reference, Version 6, First Edition*, pages 57-79).

In the INIT section, get the current help level and activate the correct choice group. The first menu may extract the help level from the user profile. After fetching the value, you will need to set the value of macro variable HELPLEV:

```
call symput('helplev',
            helplev);
rc=activate('helplev',1);
```

In subsequent windows, you can use:

```
helplev = symget('helplev');
rc=activate('helplev',1);
```

In the MAIN section, specify the following statements to reset the value of HELPLEV if users change the help level:

```
if modified(helplev) then do;
    call symput('helplev',
                helplev);
end;
```

Online documentation with text searching

Online documentation can bring with it a set of potential problems: screen formatting for online as opposed to printed information, maintenance headaches if you have to support both online and printed documentation, and the difficulty of keeping users oriented in the document. However, if you overcome these small obstacles, good online documentation offers users the opportunity to browse a complete, well-organized source of information about your application, with the added benefit of text searching capabilities.

To provide online documentation with minimal programming, we use a HELP entry to contain the document and additional PROGRAM entries for text searching and printing. (HELP entries allow you to use FIND commands.)

To create a simple online document, follow these steps:

1. Create a HELP entry and enter the text for your document (or include it from an external file). Note that the entry will display continuous text, without separate frames.
2. As the program for text searching, create a PROGRAM entry with a single field named FTEXT, forty characters long. This PROGRAM entry captures FIND commands and passes them back to the HELP entry. Use the following SCL statements:

```
length quote $ 1;
```

```
INIT:
```

```
    cursor ftext;
return;
```

```
MAIN:
```

```
    if ftext NE _blank_ then do;
        if substr(ftext,1,1)
            in (' ','"') then
            ftext= substr(
                ftext,2,(length
                    (ftext)-2));
```

```
    if index(ftext,'"') then
        quote='";
    else quote='";
```

```
        fcom='find '    ||
            quote        ||
            trim(ftext)   ||
            quote;
        call execcmd(fcom);
        _status_='H';
end;
return;

TERM:
return;
```

3. Determine how you want users to activate the text search feature. We create an action bar with the items Find and Rfind (repeat find), but you may prefer another means, such as letting users press function keys or select a field.

Whatever your choice, when users activate the Find feature, call the PROGRAM entry you created in step 2 above. (As a command to call the PROGRAM entry, use:

```
CDISPLAY
libref.catalog.entry.PROGRAM
```

For repeat finds, use the RFIND command.)

When users activate the Find feature, the PROGRAM entry appears. The user types in the text to find and presses Enter. The program brings the next line containing the text to the top of the window.

Other possible approaches: If your document would be difficult to maintain as a HELP entry because it changes over time, you can store the document in an external file and use the FSLIST procedure to display it. Users can issue search commands to locate text.

Conclusion

You can implement the principles we've suggested in many ways. We've shown just a few. The key is to organize from the beginning of your development cycle.

Start by developing a toolkit of useful, reliable features. By establishing techniques and designing applications with them in mind, you'll provide better user support than if you tack on features at the end of a project. Planning ahead results in less superficial, more easily developed user support – and satisfied users.