ABSTRACT

Pull-down menus are a key component in creating "user-friendly" interactive SAS® software applications. This paper will help you become a more effective pmenu developer by showing you how to:

- implement "pseudo commands", which supplement or replace regular SAS commands. These include, but are not limited to, custom commands.
- assign pmenus to a variety of windows with maximum flexibility. This process is not as straightforward as you might assume.

INTRODUCTION

Pull-down menus - or pmenus as they are called in the SAS system - are an alternative to the traditional command prompt, allowing users to issue commands without typing. Popularized by PC software, pmenus have become accepted as an integral part of a "user-friendly" computer environment. They require less memorization, reduce the number of errors, and facilitate the user’s interaction with the software, especially if a mouse is available.

The SAS system provides default pmenus for all windows. You may want to replace those pmenus with your own custom pmenus, for at least three reasons:

- Not everyone likes the default pmenus. Some people, for example, find the words LOCALS and GLOBALS totally baffling.
- You may want to conceal certain commands from your users. For example, you might not want them to access the Program Editor window.
- Your application may require special functionality, which you would like users to access via pmenus.

In order to develop effective pmenus, you need to learn three techniques:

- How to use PROC PMENU
  PROC PMENU is a base product procedure which creates pmenus. In order to help people who may not be familiar with this procedure, I will include an example of a simple pmenu and the corresponding PROC PMENU source code.
- How to use pseudo commands
  The term "pseudo command" is my own invention. Such commands can be implemented using four related techniques: aliases, custom commands, CDISPLAY, and macro-based commands. These techniques give you tremendous power to expand (or restrict) the commands which you provide to users on pmenus. This section of the paper explains how to use these techniques, and the situations for which they are best suited. Once you have used pseudo commands, you will wonder how you ever managed without them.
- How to associate pmenus with windows
  There are several ways in which custom pmenus may be assigned to windows. This section of the paper describes these assignment methods, and provides usage guidelines. As you develop pmenus for an application, you will find this to be a valuable reference document.

PROC PMENU

PROC PMENU allows you to build pmenus consisting of an action bar, pull-down menus, and dialog boxes. When the source code executes, catalog entries of type PMENU are created (or modified).

Suppose you want to create a pmenu called SIMPLE and store it in MYLIB.MYCATH. The action bar of SIMPLE consists of two pull-down menus, as follows:

<table>
<thead>
<tr>
<th>EXIT</th>
<th>SCROLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>END</td>
<td>TOP</td>
</tr>
<tr>
<td>CANCEL</td>
<td>BACKWARD</td>
</tr>
<tr>
<td></td>
<td>FORWARD</td>
</tr>
<tr>
<td></td>
<td>BOTTOM</td>
</tr>
</tbody>
</table>
The source code to create this pmenu is:

```
PROC PMENU CAT=MYLIB.MYCAT;
  MENU SIMPLE;
    ITEM 'EXIT' MENU=MENU1;
    ITEM 'SCROLL' MENU=MENU2;
    MENU MENU1;
    ITEM END;
    ITEM CANCEL;
    MENU MENU2;
    ITEM TOP;
    ITEM BACKWARD;
    ITEM FORWARD;
    ITEM BOTTOM;
RUN;
```

Pmenus can, of course, be more complex and powerful, as illustrated by some of the examples in this paper. You will probably want to have multiple pmenus for different window types and situations. Chapter 26 of SAS Procedures Guide, Version 6, 3rd Edition provides a thorough presentation of PROC PMENU syntax.

**PSEUDO COMMANDS**

In the above example, all six commands which the user could select were native SAS system commands. Sometimes, however, the native commands available to a particular window are not adequate or acceptable. You can use four additional tools - aliases, custom commands, CDISPLAY, and macro-based commands - to implement "pseudo commands".

**Aliases**

An alias is simply a pmenu action bar or menu item with a different name than the native command. The mapping is made in the PROC PMENU code. For example, the following code:

```
PROC PMENU CAT=libref.catname;
  MENU menuname;
    ITEM 'RUN' SELECTION=RUN;
    SELECTION RUN 'END';
RUN;
```

associates the action bar item RUN with the command END.

An alias is helpful when a native command functions acceptably, but the command name may not be intuitive to some users.

On the other hand, aliases cannot be issued from the command prompt. If some of your users prefer to type commands at the command prompt, you should use custom commands.

**Custom commands**

You can direct a Screen Control Language (SCL) program to intercept commands before they are passed to the window's command processor. The SCL program then processes these "custom" commands according to your directions. The technique is explained in chapter 13 of SAS Screen Control Language: Usage, Version 6, First Edition.

Custom commands have a number of advantages:

- They can provide functionality which native commands do not support.
- They can be issued from a pmenu, function key, or command prompt.
- It is possible to intercept, and either redefine or reject, some (but not all) native commands.\(^1\)
- Since custom commands are processed by SCL, their meaning can be defined when the user executes the application, rather than when you compile the SCL or pmenu code. This late (run-time) binding means that the action taken in response to a particular custom command can depend on the value of a macro or window variable, for example. This makes custom commands very powerful.

But you need to be aware of the following:

- Custom commands require SCL, so you cannot allow the user to issue a custom command from a window which does not support SCL (e.g., a HELP entry).
- Every window which needs to accept a custom command must contain the code to process it. If you support custom commands in many entries, you could write a lot of repetitive code. To avoid this and minimize maintenance problems, place this code in SCL macros. These are macros which contain SCL statements. They can be placed in autocall libraries, and will execute when the SCL program is compiled. Compile time and memory usage will increase slightly, but SCL execution time is not affected.
Custom command processing requires you to use at least the ENTER option of the CONTROL statement. This means that the MAIN block of your SCL program will execute whenever the user presses the ENTER key, regardless of whether any fields were modified. You will have to adjust your SCL programming logic to account for this.

A hidden gem: CDISPLAY

There is a command called CDISPLAY - which you can abbreviate as CDI - which can invoke a PROGRAM, SCL, HELP, MENU, or CBT entry from any SAS/AF® or SAS/FSP® display window. This command was apparently developed for internal SAS Institute use, and is undocumented. The syntax of this command is:

```
CDI libref.catname.entryname.entrytype
```

When the user exits from the CDISPLAYed entry (or its execution stream), control returns to the window from which the CDI command was issued.

This technique has great flexibility:

- Like custom commands, CDI can provide functionality which native commands do not support, and is ideal for a pmenu or function key. CDI can be issued from a command prompt. This is not practical, however, since the user would have too much to remember and type.
- Since CDI is a native command, you need no SCL in the calling program. This makes your work easier, and makes this kind of pseudo command available from windows which have no SCL program.
- You must decide the name of the CDISPLAYed catalog entry when the pmenu is created (when PROC PMENU code is submitted). At first glance, this early (compile-time) binding would seem to be a handicap. However, the CDISPLAYed entry may itself contain SCL, and thus be able to function differently based on run-time conditions.

For example, suppose a user selects an item entitled SESSION INFO from the pmenu on the display window of entry X.HELP. The information to be shown depends on whether the user is a beginner or advanced. The corresponding PROC PMENU code is:

```
PROC PMENU CAT=libref.catname;
  MENU menuName;
    ITEM 'SESSION INFO' SELECTION=St1;
      SELECTION SI 'CDI MYLIB.MYCats.SI.PROGRAM';
    RUN;

SI.PROGRAM then determines the user's status by table lookup, displays appropriate information, and returns control to X.HELP.
```

Macro-based commands

It is a little-known fact that macros will execute and global macro variables will resolve when issued as commands. If the resulting code is either a native or a pseudo command, that command will then execute.

This can be a convenience:

- when you want to change details of a command without modifying all the PROC PMENU code which refers to that command.
- when the behavior of a command needs to be conditional on global macro variables, whose values may change during a session.

ASSOCIATING PMENUS WITH WINDOWS

Associating pmenus with windows is more complex than one would think. The technique you choose will depend on the window type, how the window is invoked, and the number of catalogs used in the application.

The following techniques will be discussed:

- using general attributes
- using the SCL PMENU function
- using the SETPMENU command
- handling exceptional situations

Using general attributes

There is a single, 8-character field entitled COMMAND MENU on the General Attributes (GATTR) window of PROGRAM, MENU, HELP, and CBT entries. This allows you to type the entry name of a pmenu, but not its libref or catalog name. If you use this technique, therefore, the pmenu must be stored in the same catalog as the PROGRAM entry.
This is the easiest way to assign a pmenu to a window. It is, however, limited to SAS/AF entries and (for practical purposes) to single-catalog applications.

**Using the SCL PMENU function**

There are several situations in which the SCL PMENU function is preferable to general attributes.

- The best design for a large, complex SAS/AF application may involve several catalogs. The GATTR method of assigning pmenus does not work well in such situations, because you cannot identify the library and catalog.

You could store duplicate copies of the pmenus in each catalog, but this is inefficient and poor maintenance policy.

You might assume that you can store all pmenus in a single catalog, and direct the SCL SEARCH function to find the pmenu in the catalog search path. Unfortunately, SEARCH does not function this way.

The best solution for PROGRAM entries in multi-catalog applications is to use the PMENU function in the SCL program. This function will accept a full 3-level name, allowing you to store the pmenus wherever you want:

\[
rc = \text{PMENU('libref.catname.entryname');}
\]

- The PMENU function is also useful for SAS/FSP windows.

By analogy to SAS/AF, you might expect to find a COMMAND MENU field on the FSEDIT/FSBROWSE General Parameters window in screen modification mode. Unfortunately, there is no such field.

Regardless of the number of catalogs in your application, therefore, you should use the PMENU function. Call it in the FSEINIT block of the SCL program.

There is neither a GATTR nor a General Parameters window for FSVIEW, but you can add a PMENU function call to any FSVIEW formula. For example, if you have a formula for variable \( x \):

\[
x = y + z; \\
rc = \text{PMENU('libref.catname.entryname');}
\]

**SETPMENU command**

What do you do if a window has no SCL program? Some windows, such as MENU, HELP, CBT, and FSLIST, simply cannot have an SCL program. Furthermore, you may choose not to write an SCL program for windows such as FSEDIT, FSBROWSE, and FSVIEW. In such situations you will be unable to use the PMENU function. If the window to which you want to assign the pmenu - which I will call the target window - is invoked from an SCL program, however, you can use a barely documented command called SETPMENU

The general approach is to stack the SETPMENU command. "Stacking" means that you are requesting deferred execution of the command - in this case until the SCL program returns control to the user. The SCL tool for stacking a command is the EXECCMD routine.

When the target is a SAS/AF window, use the following code in the calling program:

```
CALL EXECCMD('SETPMENU libref.catname.entryname'); 
CALL DISPLAY('libref.catname.entryname.entrytype');
```

This approach will work for FSEDIT, FSBROWSE, and FSLIST windows as well:

```
CALL EXECCMD('SETPMENU libref.catname.entryname'); 
CALL EXECCMD('SETPMENU libref.catname.entryname'); 
CALL FSEDIT('libref.catname.entryname'); 
CALL FSUST('fileref');
```

For some reason, stacking SETPMENU before CALL FSVIEW does not work properly. Instead, stack the SAS Display Manager System FSVIEW window command, followed by SETPMENU:

```
CALL EXECCMD('FSVIEW libref.datafile 
libref.catname.entryname;  
SETPMENU libref.catname.entryname');
```

Native selection list windows are displayed by invoking special SCL functions such as DIRLIST. To attach pmenus to such windows, you must also use SETPMENU:

```
CALL EXECCMD('SETPMENU libref.catname.entryname'); 
list = DIRLIST('...');
```
Note that when you use SCL (as opposed to general attributes) to assign pmenus to windows, different pmenus may be displayed to the user based on runtime conditions.

Handling exceptional situations

There are some situations which cannot be handled by the above techniques.

SAS/AF and SAS/FSP windows

Consider the following scenarios:

- **Problem:** The user issues a HELP command, and you specified the entry where help resides via the HELP fields on a GATTR window, and that entry is in a different catalog than the pmenu.
  
  **Solution:** Intercept and redefine the HELP command using custom command processing and CONTROL ALLCMDs.

- **Problem:** The user submits PROC FSEDIT from the Program Editor, without specifying a custom screen.
  
  **Solution:** None, short of making an alternate SASHELP library (see below).

- **Problem:** You have a macro which invokes PROC FSLIST and which is submitted from an SCL SUBMIT block. It is impractical to replace the entire macro with an SCL program.
  
  **Solution:** Rewrite the macro, replacing PROC FSLIST with PROC DISPLAY pointing to an SCL entry. Inside the SCL program, stack SETPMENU and call the FSLIST window.

Display manager windows

The default pmenus for display manager windows reside in SASHELP catalogs, and you cannot get display manager to search for pmenus beyond the SASHELP library. At this time, the only way to substitute your own pmenus for the default ones is to copy the entire SASHELP library - which is huge - into another library, replace the pmenus, and assign the libref SASHELP to the new library at invocation time. This wastes disk space and is tedious, especially since you would have to recopy every time there is a software upgrade.

I hope that SAS Institute will recognize that there is a legitimate need to customize display manager pmenus. Until then, I recommend that you provide training on the default pmenus to users who need display manager access, and block other users from accessing display manager in the first place.

DATA step and macro windows

To assign custom pmenus to DATA step and macro windows, use the MENUS option on the WINDOW or %window statements.

PROC REPORT window

To assign custom pmenus to the report and compute windows of PROC REPORT, specify libref, catalog, and pmenu entry names in a profile entry. Point to the this entry using the PROFILE= option in PROC REPORT.

CONCLUSION

Pmenus are a powerful tool in creating user-friendly interactive applications. You will need knowledge and experience, however, to use pmenus to their full potential. This paper has attempted to assist you by explaining how to use PROC PMENU, issue pseudo commands, and associate pmenus with various SAS system windows.

ACKNOWLEDGMENTS

I would like to thank:

- SAS Institute Technical Support for providing information about display manager pmenus and the status of the CDISPLAY command.
- Dan Gronell of Pfizer Inc. for telling me about CDISPLAY in the first place.
- my wife, Paulette Staum, for researching some of the more complex techniques described above, and for her insightful suggestions for improving this paper.
To contact the author, write to:

Roger Staum
American Cyanamid Company
Building 60, Room 311B
North Middletown Road
Pearl River, NY 10965

SAS, SAS/AF, and SAS/FSP are registered trademarks of SAS Institute Inc. in the USA and other countries. MVS and AIX are registered trademarks of IBM Corp. WINDOWS is a registered trademark of Microsoft Corp. © indicates USA registration.

1 See the ALLCMDS option of the CONTROL statement in SAS Technical Report P-216, chapters 4 and 20.
2 I cannot guarantee that CDI SP L will work on all platforms and versions of the SAS system, but we have tested it successfully on v. 6.07 under MVS® and AIX®, as well as on the β-test v. 6.08 under WINDOWS®.
3 The SASware ballot contains an item requesting that this attribute be increased to 3 fields.
4 This function is new to version 6.07. See SAS Technical Report P-216, pp. 249-250.
5 You can find a description on p. 419 of SAS/FSP Software: Usage and Reference, v. 6, First Ed. It is incorrectly identified, however, as the command PMENUNAME.
6 For example, the macro is called frequently and/or contains a lot of other code which must be submitted.
7 PROC DISPLAY, by the way, is how you invoke a SAS/AF application outside display manager mode.
8 You cannot issue a LIBNAME statement for SASHELP in the middle of a SAS session.