CHAPTER
27

The PMENU Procedure

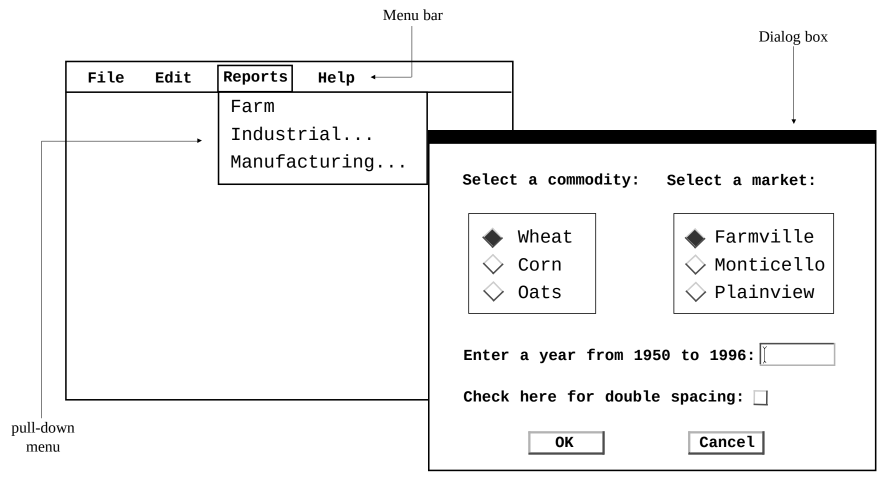
Overview

The PMENU procedure defines menus that can be used in DATA step windows, macro windows, both SAS/AF and SAS/FSP windows, or in any SAS application that enables you to specify customized menus.

Menus can replace the command line as a way to execute commands. To activate menus, issue the PMENU command from any command line. *Menus must be activated in order for them to appear.*

When menus are activated, each active window has a *menu bar*, which lists items that you can select. Depending upon which item you select, SAS either processes a command, displays a menu or a submenu, or requests that you complete information in a *dialog box*. The *dialog box* is simply a box of questions or choices that require answers before an action can be performed. Figure 27.1 on page 746 illustrates features that you can create with PROC PMENU.

Figure 27.1 Menu Bar, Pull-Down Menu, and Dialog Box



Note: A menu bar in some operating environments may appear as a popup menu or may appear at the bottom of the window. △

The PMENU procedure produces no immediately visible output. It simply builds a catalog entry of type PMENU that can be used later in an application.

Procedure Syntax

Restriction: You must use at least one MENU statement followed by at least one ITEM statement.

Tip: Supports RUN group processing

Tip: Supports Output Delivery System (see Chapter 2, "Fundamental Concepts for Using Base SAS Procedures," on page 15)

Reminder: You can also use appropriate global statements with this procedure. See Chapter 2, "Fundamental Concepts for Using Base SAS Procedures," on page 15 for a list.

PROC PMENU <CATALOG=<*libref.*>*catalog*>
  <DESC '*entry-description*'>;

  MENU *menu-bar*;
    ITEM *command* <*option(s)*>;
    ITEM *'menu-item'* <*option(s)*>;
      DIALOG *dialog-box 'command-string*
        field-number-specification';
        CHECKBOX <ON> *#line @column*
          'text-for-selection'
          <COLOR=*color*> <SUBSTITUTE=*'text-for-substitution'*>;
        RADIOBOX DEFAULT=*button-number*;
          RBUTTON <NONE> *#line @column*
            'text-for-selection' <COLOR=*color*>
            <SUBSTITUTE=*'text-for-substitution'*>;
        TEXT *#line @column field-description*

<ATTR=*attribute*> <COLOR=*color*>;
MENU *pull-down-menu*;
SELECTION*selection 'command-string'*;
SEPARATOR;
SUBMENU *submenu-name SAS-file*;

To do this	Use this statement
Define choices a user can make in a dialog box	CHECKBOX
Describe a dialog box that is associated with an item in a pull-down menu	DIALOG
Identify an item to be listed in a menu bar or in a pull-down menu	ITEM
Name the catalog entry or define a pull-down menu	MENU
List and define mutually exclusive choices within a dialog box	RADIOBOX and RBUTTON
Define a command that is submitted when an item is selected	SELECTION
Draw a line between items in a pull-down menu	SEPARATOR
Define a common submenu associated with an item	SUBMENU
Specify text and the input fields for a dialog box	TEXT

PROC PMENU Statement

Invokes the PMENU procedure and specifies where to store all PMENU catalog entries created in the PROC PMENU step.

PROC PMENU <CATALOG=<*libref.*>*catalog*>
<DESC *'entry-description'*>;

Options

CATALOG=<*libref.*>*catalog*
specifies the catalog in which you want to store PMENU entries.

Default: If you omit *libref*, the PMENU entries are stored in a catalog in the SASUSER data library. If you omit CATALOG=, the entries are stored in the SASUSER.PROFILE catalog.

Featured in: Example 1 on page 762

DESC *'entry-description'*
provides a description for the PMENU catalog entries created in the step.

Default: `Menu description`

Note: These descriptions are displayed when you use the CATALOG window in the windowing environment or the CONTENTS statement in the CATALOG procedure. △

CHECKBOX Statement

Defines choices that a user can make within a dialog box.

Restriction: Must be used after a DIALOG statement.

CHECKBOX <ON> *#line @column*
'text-for-selection'
<COLOR=*color*> <SUBSTITUTE=*'text-for-substitution'*>;

Required Arguments

column
specifies the column in the dialog box where the checkbox and text are placed.

line
specifies the line in the dialog box where the checkbox and text are placed.

text-for-selection
defines the text that describes this check box. This text appears in the window and, if the SUBSTITUTE= option is not used, is also inserted into the command in the preceding DIALOG statement when the user selects the check box.

Options

COLOR=*color*
defines the color of the check box and the text that describes it.

ON
indicates that by default this check box is active. If you use this option, you must specify it immediately after the CHECKBOX keyword.

SUBSTITUTE=*'text-for-substitution'*
specifies the text that is to be inserted into the command in the DIALOG statement.

Check Boxes in a Dialog Box

Each CHECKBOX statement defines a single item that the user can select independent of other selections. That is, if you define five choices with five CHECKBOX statements, the user can select any combination of these choices. When the user selects choices, the *text-for-selection* values that are associated with the selections are inserted into the command string of the previous DIALOG statement at field locations prefixed by an ampersand (&).

DIALOG Statement

Describes a dialog box that is associated with an item on a pull-down menu.

Restriction: Must be followed by at least one TEXT statement.

Featured in: Example 2 on page 764, Example 3 on page 767, and Example 4 on page 773

DIALOG *dialog-box* '*command-string*
field-number-specification';

Required Arguments

command-string
is the command or partial command that is executed when the item is selected. The limit of the *command-string* that results after the substitutions are made is the command-line limit for your operating environment. Typically, the command-line limit is approximately 80 characters.
The limit for '*command-string field-number-specification*' is 200 characters.

Note: If you are using PROC PMENU to submit any command that is valid only in the PROGRAM EDITOR window (such as the INCLUDE command), you must have the windowing environment running, and you must return control to the PROGRAM EDITOR window. △

dialog-box
is the same name specified for the DIALOG= option in a previous ITEM statement.

field-number-specification
can be one or more of the following:

@1...@*n*

%1...%*n*

&1...&*n*
You can embed the field numbers, for example @1, %1, or &1, in the command string and mix different types of field numbers within a command string. The numeric portion of the field number corresponds to the relative position of TEXT, RADIOBOX, and CHECKBOX statements, not to any actual number in these statements.

@1...@*n*
are optional TEXT statement numbers that can add information to the command before it is submitted. Numbers preceded by an at sign (@) correspond to TEXT statements that use the LEN= option to define input fields.

%1...%*n*
are optional RADIOBOX statement numbers that can add information to the command before it is submitted. Numbers preceded by a percent sign (%) correspond to RADIOBOX statements following the DIALOG statement.

Note: Keep in mind that the numbers correspond to RADIOBOX statements, not to RBUTTON statements. △

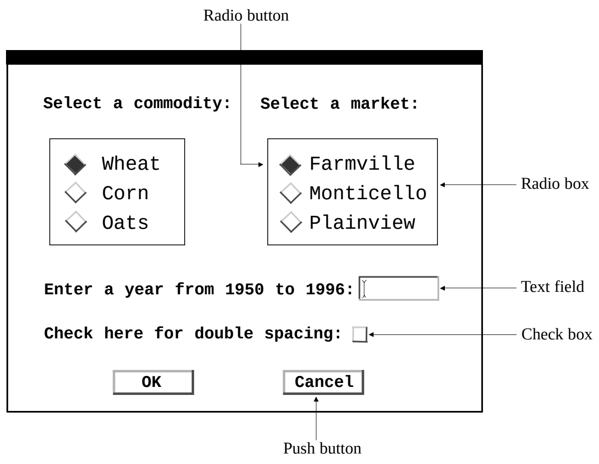
&1...&*n*
are optional CHECKBOX statement numbers that can add information to the command before it is submitted. Numbers preceded by an ampersand (&) correspond to CHECKBOX statements following the DIALOG statement.

Note: To specify a literal @ (at sign), % (percent sign), or & (ampersand) in the *command-string*, use a double character: @@ (at signs), %% (percent signs), or && (ampersands). △

Details

- You cannot control the placement of the dialog box. The dialog box is not scrollable. The size and placement of the dialog box are determined by your windowing environment.
- To use the DIALOG statement, specify an ITEM statement with the DIALOG= option in the ITEM statement.
- The ITEM statement creates an entry in a menu bar or in a pull-down menu, and the DIALOG= option specifies which DIALOG statement describes the dialog box.
- You can use CHECKBOX, RADIOBOX, and RBUTTON statements to define the contents of the dialog box.
- Figure 27.2 on page 750 shows a typical dialog box. A dialog box can request information in three ways:
  - Fill in a field. Fields that accept text from a user are called *text fields*.
  - Choose from a list of mutually exclusive choices. A group of selections of this type is called a *radio box*, and each individual selection is called a *radio button*.
  - Indicate whether you want to select other independent choices. For example, you could choose to use various options by selecting any or all of the listed selections. A selection of this type is called a *check box*.

Figure 27.2 A Typical Dialog Box



Dialog boxes have two or more *push buttons*, such as OK and Cancel, automatically built into the box.* A *push button* causes an action to occur.

ITEM Statement

Identifies an item to be listed in a menu bar or in a pull-down menu.

Featured in: Example 1 on page 762

* The actual names of the push buttons vary in different windowing environments.

ITEM *command <option(s)><action-options>*;

ITEM *'menu-item' <option(s)><action-options>*;

To do this	Use this option
Specify the action for the item	
Associate the item with a dialog box	DIALOG=
Associate the item with a pull-down menu	MENU=
Associate the item with a command	SELECTION=
Associate the item with a common submenu	SUBMENU=
Specify help text for an item	HELP=
Define a key that can be used instead of the pull-down menu	ACCELERATE=
Indicate that the item is not an active choice in the window	GRAY
Provide an ID number for an item	ID=
Define a single character that can select the item	MNEMONIC=
Place a check box or a radio button next to an item	STATE=

Required Arguments

command
a single word that is a valid SAS command for the window in which the menu appears. Commands that are more than one word, such as WHERE CLEAR, must be in single quotes. The *command* appears in uppercase letters on the menu bar.

If you want to control the casing of a SAS command on the menu, enclose the command in single quotes and the case that you used then appears on the menu.

menu-item
a word or text string, enclosed in quotes, that describes the action that occurs when the user selects this item. A menu item should not begin with a percent sign (%).

Options

ACCELERATE=*name-of-key*
defines a key sequence that can be used instead of selecting an item. When the user presses the key sequence, it has the same effect as selecting the item from the menu bar or pull-down menu.

Restriction: The functionality of this option is limited to only a few characters. For details, see the SAS documentation for your operating environment.

Restriction: This option is not available in all operating environments. If you include this option and it is not available in your operating environment, the option is ignored.

action-option
is one of the following:

DIALOG=*dialog-box*
the name of an associated DIALOG statement, which displays a dialog box when the user selects this item.

Featured in: Example 3 on page 767

MENU=*pull-down-menu*
the name of an associated MENU statement, which displays a pull-down menu when the user selects this item.

Featured in: Example 1 on page 762

SELECTION=*selection*
the name of an associated SELECTION statement, which submits a command when the user selects this item.

Featured in: Example 1 on page 762

SUBMENU=*submenu*
the name of an associated SUBMENU statement, which displays a pmenu entry when the user selects this item.

Featured in: Example 1 on page 762

If no DIALOG=, MENU=, SELECTION=, or SUBMENU= option is specified, the *command* or *menu-item* text string is submitted as a command-line command when the user selects the item.

GRAY
indicates that the item is not an active choice in this window. This option is useful when you want to define standard lists of items for many windows, but not all items are valid in all windows. When this option is set and the user selects the item, no action occurs.

HELP='*help-text*'
specifies text that is displayed when the user displays the menu item. For example, if you use a mouse to pull down a menu, hold the mouse button on the item and the text is displayed.

Restriction: This option is not available in all operating environments. If you include this option and it is not available in your operating environment, the option is ignored.

Tip: The place where the text is displayed is operating environment-specific.

ID=*integer*
a value that is used as an identifier for an item in a pull-down menu. This identifier is used within a SAS/AF application to selectively gray or ungray items in a menu or to set the state of an item as a check box or a radio button.

Minimum: 3001

Restriction: Integers from 0 - 3000 are reserved for operating environment and SAS System use.

Restriction: This option is not available in all operating environments. If you include this option and it is not available in your operating environment, the option is ignored.

Tip: ID= is useful with the WINFO function in SAS Screen Control Language.

Tip: You can use the same ID for more than one item.

See also: STATE= option on page 753

MNEMONIC=*character*
underlines the first occurrence of *character* in the text string that appears on the pull-down menu. The *character* must be in the text string.

The *character* is typically used in combination with another key, such as ALT. When you use the key sequence, it has the same effect as putting your cursor on the item. But it *does not* invoke the action that the item controls.

Restriction: This option is not available in all operating environments. If you include this option and it is not available in your operating environment, the option is ignored.

STATE=CHECK|RADIO
provides the ability to place a check box or a radio button next to an item that has been selected.

Tip: STATE= is used with the ID= option and the WINFO function in SAS Screen Control Language.

Restriction: This option is not available in all operating environments. If you include this option and it is not available in your operating environment, the option is ignored.

Defining Items on the Menu Bar

You must use ITEM statements to name all the items that appear in a menu bar. You also use the ITEM statement to name the items that appear in any pull-down menus. The items that you specify in the ITEM statement can be commands that are issued when the user selects the item, or they can be descriptions of other actions that are performed by associated DIALOG, MENU, SELECTION, or SUBMENU statements.

All ITEM statements for a menu must be placed immediately after the MENU statement and before any DIALOG, SELECTION, SUBMENU, or other MENU statements. In some operating environments, you can insert SEPARATOR statements between ITEM statements to produce lines separating groups of items in a pull-down menu. See "SEPARATOR Statement" on page 757 for more information.

CAUTION:
If you specify a menu bar that is too long for the window, it might be truncated or wrapped to multiple lines. △

MENU Statement

Names the catalog entry that stores the menus or defines a pull-down menu.

Featured in: Example 1 on page 762

MENU *menu-bar*;

MENU *pull-down-menu*;

Required Arguments

One of the following arguments is required:

menu-bar
names the catalog entry that stores the menus.

pull-down-menu
names the pull-down menu that appears when the user selects an item in the menu bar. The value of *pull-down-menu* must match the *pull-down-menu* name that is specified in the MENU= option in a previous ITEM statement.

Defining Pull-Down Menus

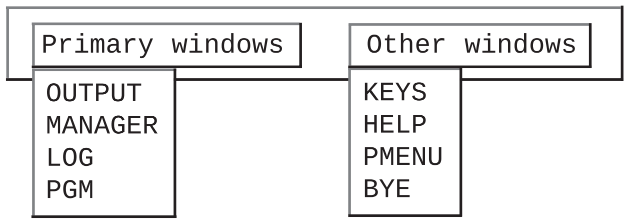
When used to define a pull-down menu, the MENU statement must follow an ITEM statement that specifies the MENU= option. Both the ITEM statement and the MENU statement for the pull-down menu must be in the same RUN group as the MENU statement that defines the menu bar for the PMENU catalog entry.

For both menu bars and pull-down menus, follow the MENU statement with ITEM statements that define each of the items that appear on the menu. Group all ITEM statements for a menu together. For example, the following PROC PMENU step creates one catalog entry, WINDOWS, which produces a menu bar with two items, **`Primary windows`** and **`Other windows`**. When you select one of these items, a pull-down menu is displayed.

```
libname proclib 'SAS-data-library';

proc pmenu cat=proclib.mycat;

      /* create catalog entry */
   menu windows;
   item 'Primary windows' menu=prime;
   item 'Other windows' menu=other;

      /* create first pull-down menu */
   menu prime;
   item output;
   item manager;
   item log;
   item pgm;

      /* create second pull-down menu */
   menu other;
   item keys;
   item help;
   item pmenu;
   item bye;

   /* end of run group */
run;
```

Figure 27.3 on page 755 shows the resulting menu selections.

Figure 27.3 Pull-Down Menu



RADIOBOX Statement

Defines a box that contains mutually exclusive choices within a dialog box.

Restriction: Must be used after a DIALOG statement.

Restriction: Must be followed by one or more RBUTTON statements.

Featured in: Example 3 on page 767

RADIOBOX DEFAULT=*button-number*;

Required Arguments

DEFAULT=*button-number*
indicates which radio button is the default.

Default: 1

Details

The RADIOBOX statement indicates the beginning of a list of selections. Immediately after the RADIOBOX statement, you must list an RBUTTON statement for each of the selections the user can make. When the user makes a choice, the text value that is associated with the selection is inserted into the command string of the previous DIALOG statement at field locations prefixed by a percent sign (%).

RBUTTON Statement

Lists mutually exclusive choices within a dialog box.

Restriction: Must be used after a RADIOBOX statement.

Featured in: Example 3 on page 767

RBUTTON <NONE> *#line @column*
'text-for-selection' <COLOR=*color*> <SUBSTITUTE=*'text-for-substitution'*>;

Required Arguments

column
specifies the column in the dialog box where the radio button and text are placed.

line
specifies the line in the dialog box where the radio button and text are placed.

text-for-selection
defines the text that appears in the dialog box and, if the SUBSTITUTE= option is not used, defines the text that is inserted into the command in the preceding DIALOG statement.

CAUTION:
Be careful not to overlap columns and lines when placing text and radio buttons. You receive an error message if you overlap text or buttons. In addition, specify space between other text and a radio button. △

Options

COLOR=*color*
defines the color of the radio button and the text that describes the button.

Restriction: This option is not available in all operating environments. If you include this option and it is not available in your operating environment, the option is ignored.

NONE
defines a button that indicates none of the other choices. Defining this button enables the user to ignore any of the other choices. No characters, including blanks, are inserted into the DIALOG statement.

Restriction: If you use this option, it must occur immediately after the RBUTTON keyword.

SUBSTITUTE=*'text-for-substitution'*
specifies the text that is to be inserted into the command in the DIALOG statement.

Featured in: Example 3 on page 767

SELECTION Statement

Defines a command that is submitted when an item is selected.

Restriction: Must be used after an ITEM statement

Featured in: Example 1 on page 762 and Example 4 on page 773

SELECTION *selection 'command-string'*;

Required Arguments

selection
is the same name specified for the SELECTION= option in a previous ITEM statement.

command-string
is a text string, enclosed in quotes, that is submitted as a command-line command when the user selects this item. There is a limit of 200 characters for *command-string*. However, the command-line limit of approximately 80 characters cannot be exceeded. The command-line limit differs slightly for various operating environments.

Details

You define the name of the item in the ITEM statement and specify the SELECTION= option to associate the item with a subsequent SELECTION statement. The SELECTION statement then defines the actual command that is submitted when the user chooses the item in the menu bar or pull-down menu.

You are likely to use the SELECTION statement to define a command string. You create a simple alias by using the ITEM statement, which invokes a longer command string that is defined in the SELECTION statement. For example, you could include an item in the menu bar that invokes a WINDOW statement to allow data entry. The actual commands that are processed when the user selects this item are the commands to include and submit the application.

Note: If you are using PROC PMENU to issue any command that is valid only in the PROGRAM EDITOR window (such as the INCLUDE command), you must have the windowing environment running, and you must return control to the PROGRAM EDITOR window. △

SEPARATOR Statement

Draws a line between items on a pull-down menu.

Restriction: Must be used after an ITEM statement.

Restriction: Not available in all operating environments.

SEPARATOR;

SUBMENU Statement

Specifies the SAS file that contains a common submenu associated with an item.

Featured in: Example 1 on page 762

SUBMENU *submenu-name SAS-file*;

Required Arguments

submenu-name
specifies a name for the submenu statement. To associate a submenu with a menu item, *submenu-name* must match the submenu name specified in the SUBMENU= action-option in the ITEM statement.

SAS-file
specifies the name of the SAS file that contains the common submenu.

TEXT Statement

Specifies text and the input fields for a dialog box.

Restriction: Can be used only after a DIALOG statement.

Featured in: Example 2 on page 764

TEXT *#line @column field-description*
    <ATTR=*attribute*> <COLOR=*color*>;

Required Arguments

column
specifies the starting column for the text or input field.

field-description
defines how the TEXT statement is used. The *field-description* can be one of the following:

LEN=*field-length*
is the length of an input field in which the user can enter information. If the LEN= argument is used, the information entered in the field is inserted into the command string of the previous DIALOG statement at field locations prefixed by an at sign (@).

Featured in: Example 2 on page 764

'*text*'
is the text string that appears inside the dialog box at the location defined by *line* and *column*.

line
specifies the line number for the text or input field.

Options

ATTR=*attribute*
defines the attribute for the text or input field. Valid attribute values are

- □ BLINK
- □ HIGHLIGH
- □ REV_VIDE
- □ UNDERLIN

Restriction: This option is not available in all operating environments. If you include this option and it is not available in your operating environment, the option is ignored.

Restriction: Your hardware may not support all of these attributes.

COLOR=*color*
defines the color for the text or input field characters. These are the color values that you can use:

BLACK	BROWN
GRAY	MAGENTA
PINK	WHITE
BLUE	CYAN
GREEN	ORANGE
RED	YELLOW

Restriction: This option is not available in all operating environments. If you include this option and it is not available in your operating environment, the option is ignored.

Restriction: Your hardware may not support all of these colors.

Concepts

Procedure Execution

You can define multiple menus by separating their definitions with RUN statements. A group of statements that ends with a RUN statement is called a *RUN group*. You must completely define a PMENU catalog entry before submitting a RUN statement. You do not have to restart the procedure after a RUN statement.

You must include an initial MENU statement that defines the menu bar, and you must include all ITEM statements and any SELECTION, MENU, SUBMENU, and DIALOG statements as well as statements that are associated with the DIALOG statement within the same RUN group. For example, the following statements define two separate PMENU catalog entries. Both are stored in the same catalog, but each PMENU catalog entry is independent of the other. In the example, both PMENU catalog entries create menu bars that simply list windowing environment commands the user can select and execute:

```
libname proclib 'SAS-data-library';

proc pmenu catalog=proclib.mycat;
```

```
      menu menu1;
      item end;
      item bye;
   run;

      menu menu2;
      item end;
      item pgm;
      item log;
      item output;
   run;
```

When you submit these statements, you receive a message that says that the PMENU entries have been created. To display one of these menu bars, you must associate the PMENU catalog entry with a window and then activate the window with the menus turned on, as described in "Steps for Building and Using PMENU Catalog Entries" on page 760.

Ending the Procedure

Submit a QUIT, DATA, or new PROC statement to execute any statements that have not executed and end the PMENU procedure. Submit a RUN CANCEL statement to cancel any statements that have not executed and end the PMENU procedure.

Steps for Building and Using PMENU Catalog Entries

In most cases, building and using PMENU entries requires the following steps:

1 Use PROC PMENU to define the menu bars, pull-down menus and other features that you want. Store the output of PROC PMENU in a SAS catalog.

2 Define a window using SAS/AF and SAS/FSP Software, or the WINDOW or %WINDOW statement in base SAS software.

3 Associate the PMENU catalog entry created in step 1 with a window by using one of the following:
   - the MENU= option in the WINDOW statement in base SAS software. See "Associating a Menu with a Window" on page 776.
   - the MENU= option in the %WINDOW statement in the macro facility.
   - the **Command Menu** field in the GATTR window in PROGRAM entries in SAS/AF Software.
   - the Keys, Pmenu, and Commands window in a FRAME entry in SAS/AF Software. See Example 5 on page 779.
   - the PMENU function in SAS/AF and SAS/FSP Software.
   - the SETPMENU command in SAS/FSP Software. See Example 1 on page 762.

4 Activate the window you have created. Make sure that the menus are turned on.

Templates for Coding PROC PMENU Steps

The following coding templates summarize how to use the statements in the PMENU procedure. Refer to descriptions of the statements for more information:

- Build a simple menu bar. All items on the menu bar are windowing environment commands:

```
proc pmenu;
   menu menu-bar;
   item command;
   ...more-ITEM-statements...
run;
```

□ Create a menu bar with an item that produces a pull-down menu:

```
proc pmenu;
    menu menu-bar;
    item 'menu-item' menu=pull-down-menu;
    ...more-ITEM-statements...
    menu pull-down-menu;
    ...ITEM-statements-for-pull-down-menu...
run;
```

□ Create a menu bar with an item that submits a command other than that which appears on the menu bar:

```
proc pmenu;
   menu menu-bar;
   item 'menu-item' selection=selection;
   ...more-ITEM-statements...
   selection selection 'command-string';
run;
```

□ Create a menu bar with an item that opens a dialog box, which displays information and requests text input:

```
proc pmenu;
   menu menu-bar;
   item 'menu-item' menu=pull-down-menu;
   ...more-ITEM-statements...
   menu pull-down-menu;
      item 'menu-item' dialog=dialog-box;
      dialog dialog-box 'command @1';
         text #line @column 'text';
         text #line @column LEN=field-length;
run;
```

□ Create a menu bar with an item that opens a dialog box, which permits one choice from a list of possible values:

```
proc pmenu;
   menu menu-bar;
   item 'menu-item' menu=pull-down-menu;
   ...more-ITEM-statements...
   menu pull-down-menu;
      item 'menu-item' dialog=dialog-box;
      dialog dialog-box 'command %1';
         text #line @column 'text';
         radiobox default=button-number;
         rbutton #line @column
                 'text-for-selection';
         ...more-RBUTTON-statements...
run;
```

□ Create a menu bar with an item that opens a dialog box, which permits several independent choices:

```
proc pmenu;
   menu menu-bar;
   item 'menu-item' menu=pull-down-menu;
   ...more-ITEM-statements...
   menu pull-down-menu;
      item 'menu-item' dialog=dialog-box;
      dialog dialog-box 'command &1';
         text #line @column 'text';
         checkbox #line @column 'text';
         ...more-CHECKBOX-statements...
run;
```

Examples

The windows in these examples were produced in the UNIX environment and may appear slightly different from the same windows in other operating environments.

You should know the operating environment-specific system options that can affect how menus are displayed and merged with existing SAS menus. For details, see the SAS documentation for your operating environment.

Example 1: Building a Menu Bar for an FSEDIT Application

Procedure features:

PROC PMENU statement option:

- CATALOG=

ITEM statement options:

- MENU=
- SELECTION=
- SUBMENU=

MENU statement

SELECTION statement

SUBMENU statement

This example creates a menu bar that can be used in an FSEDIT application to replace the default menu bar. The selections available on these pull-down menus do not enable end users to delete or duplicate observations.

Program

```
libname proclib 'SAS-data-library';
```

CATALOG= specifies PROCLIB.MENUCAT as the catalog that stores the menus.

```
proc pmenu catalog=proclib.menucat;
```

The MENU statement specifies PROJECT as the name of the catalog entry. The menus are stored in the catalog entry PROCLIB.MENUCAT.PROJECT.PMENU.

```
   menu project;
```

The ITEM statements specify the items for the menu bar. The value of the MENU= option is used in a subsequent MENU statement. The **`Edit`** item uses a common predefined submenu; the menus for the other items are defined in this PROC step.

```
      item 'File' menu=f;
      item 'Edit' submenu=editmnu;
      item 'Scroll' menu=s;
      item 'Help' menu=h;
```

This group of statements defines the selections available under **`File`** on the menu bar. The first ITEM statement specifies **`Goback`** as the first selection under **`File`**. The value of the SELECTION= option corresponds to the subsequent SELECTION statement, which specifies END as the command that is issued for that selection. The second ITEM statement specifies that the SAVE command is issued for that selection.

```
      menu f;
         item 'Goback' selection=g;
         item 'Save';
         selection g 'end';
```

The SUBMENU statement associates a predefined submenu that is located in the SAS file SASHELP.CORE.EDIT with the **`Edit`** item on the menu bar. The name of this SUBMENU statement is **`EDITMNU`**, which corresponds with the name in the SUBMENU= action-option in the ITEM statement for the **`Edit`** item.

```
      submenu editmnu sashelp.core.edit;
```

This group of statements defines the selections available under **`Scroll`** on the menu bar.

```
      menu s;
         item 'Next Obs' selection=n;
         item 'Prev Obs' selection=p;
         item 'Top';
         item 'Bottom';
         selection n 'forward';
         selection p 'backward';
```

This group of statements defines the selections available under **`Help`** on the menu bar. The SETHELP command specifies a HELP entry that contains user-written information for this FSEDIT application. The semicolon that appears after the HELP entry name allows the HELP command to be included in the string. The HELP command invokes the HELP entry.

```
      menu h;
         item 'Keys';
         item 'About this application' selection=hlp;
         selection hlp 'sethelp user.menucat.staffhlp.help;help';
quit;
```

Associating a Menu Bar with an FSEDIT Session

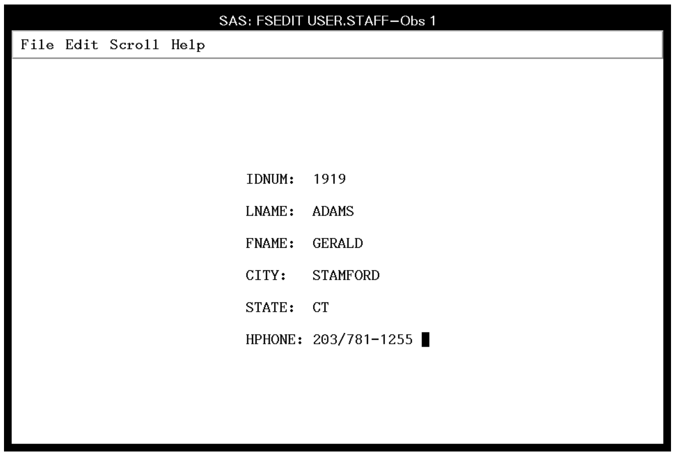
The following SETPMENU command associates the customized menu bar with the FSEDIT window.

```
setpmenu proclib.menucat.project.pmenu;pmenu on
```

You can also specify the menu bar on the command line in the FSEDIT session or with a CALL EXECCMD in Screen Control Language.

See "Associating a Menu Bar with an FSEDIT Session" on page 771 for other methods of associating the customized menu bar with the FSEDIT window.

The FSEDIT window shows the menu bar.



```
IDNUM:  1919
LNAME:  ADAMS
FNAME:  GERALD
CITY:   STAMFORD
STATE:  CT
HPHONE: 203/781-1255
```

Example 2: Collecting User Input in a Dialog Box

Procedure features:

DIALOG statement

TEXT statement option:

LEN=

This example adds a dialog box to the menus created in Example 1 on page 762. The dialog box enables the user to use a WHERE clause to subset the SAS data set.

Tasks include

- □ collecting user input in a dialog box
- □ creating customized menus for an FSEDIT application.

Program

```
libname proclib 'SAS-data-library';
```

CATALOG= specifies PROCLIB.MENUCAT as the catalog that stores the menus.

```
proc pmenu c=proclib.menucat;
```

The MENU statement specifies PROJECT as the name of the catalog entry. The menus are stored in the catalog entry PROCLIB.MENUCAT.PROJECT.PMENU.

```
   menu project;
```

The ITEM statements specify the items for the menu bar. The value of the MENU= option is used in a subsequent MENU statement.

```
      item 'File' menu=f;
      item 'Edit' menu=e;
      item 'Scroll' menu=s;
      item 'Subset' menu=sub;
      item 'Help' menu=h;
```

This group of statements defines the selections under **`File`** on the menu bar. The first ITEM statement specifies **`Go-back`** as the first selection under **`File`**. The value of the SELECTION= option corresponds to the subsequent SELECTION statement, which specifies END as the command that is issued for that selection. The second ITEM statement specifies that the SAVE command is issued for that selection.

```
      menu f;
         item 'Goback' selection=g;
         item 'Save';
         selection g 'end';
```

This group of statements defines the selections available under **`Edit`** on the menu bar.

```
      menu e;
         item 'Cancel';
         item 'Add';
```

This group of statements defines the selections available under **`Scroll`** on the menu bar.

```
      menu s;
         item 'Next Obs' selection=n;
         item 'Prev Obs' selection=p;
         item 'Top';
         item 'Bottom';
         selection n 'forward';
         selection p 'backward';
```

This group of statements defines the selections available under **`Subset`** on the menu bar. The value **`d1`** in the DIALOG= option is used in the subsequent DIALOG statement.

```
      menu sub;
         item 'Where' dialog=d1;
         item 'Where Clear';
```

This group of statements defines the selections available under **`Help`** on the menu bar. The SETHELP command specifies a HELP entry that contains user-written information for this FSEDIT application. The semicolon allows for the HELP command to be included in the string. The HELP command invokes the HELP entry.

```
      menu h;
         item 'Keys';
         item 'About this application' selection=hlp;
         selection hlp 'sethelp proclib.menucat.staffhlp.help;help';
```

The DIALOG statement builds a WHERE command. The arguments for the WHERE command are provided by user input into the text entry fields described by the three TEXT statements. The @1 notation is a placeholder for user input in the text field. The TEXT statements specify the text in the dialog box and the length of the input field.

```
      dialog d1 'where @1';
         text #2 @3 'Enter a valid WHERE clause or UNDO';
         text #4 @3 'WHERE ';
         text #4 @10 len=40;
quit;
```

Associating a Menu Bar with an FSEDIT Window

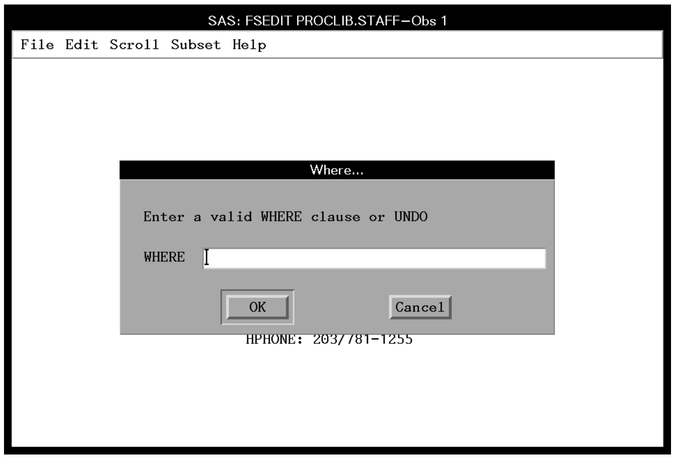
The following SETPMENU command associates the customized menu bar with the FSEDIT window.

```
setpmenu proclib.menucat.project.pmenu;pmenu on
```

You can also specify the menu bar on the command line in the FSEDIT session or with a CALL EXECCMD command in SAS Screen Control Language (SCL). Refer to *SAS Screen Control Language: Reference* for complete documentation on SCL.

See "Associating a Menu Bar with an FSEDIT Session" on page 771 for other methods of associating the customized menu bar with the FSEDIT window.

This dialog box appears when the user chooses **`Subset`** and then **`Where`**.



```
Enter a valid WHERE clause or UNDO

WHERE

HPHONE: 203/781-1255
```

Example 3: Creating a Dialog Box to Search Multiple Variables

Procedure features:
- DIALOG statement
  - SAS macro invocation
- ITEM statement
  - DIALOG= option
- RADIOBOX statement option:
  - DEFAULT=
- RBUTTON statement option:
  - SUBSTITUTE=

Other features: SAS macro invocation

This example shows how to modify the menu bar in an FSEDIT session to enable a search for one value across multiple variables. The example creates customized menus to use in an FSEDIT session. The menu structure is the same as in the preceding example, except for the WHERE dialog box.

Once selected, the menu item invokes a macro. The user input becomes values for macro parameters. The macro generates a WHERE command that expands to include all the variables needed for the search.

Tasks include

- □ associating customized menus with an FSEDIT session
- □ searching multiple variables with a WHERE clause
- □ extending PROC PMENU functionality with a SAS macro.

Program

```
libname proclib 'SAS-data-library';
```

CATALOG= specifies PROCLIB.MENUCAT as the catalog that stores the PMENU entry.

```
proc pmenu catalog=proclib.menucat;
```

The MENU statement specifies STAFF as the name of the catalog entry. The menus are stored in the catalog entry PROCLIB.MENUCAT.PROJECT.PMENU.

```
   menu project;
```

The ITEM statements specify the items for the menu bar. The value of the MENU= option is used in a subsequent MENU statement.

```
      item 'File' menu=f;
      item 'Edit' menu=e;
      item 'Scroll' menu=s;
      item 'Subset' menu=sub;
      item 'Help' menu=h;
```

This group of statements defines the selections under **`File`** on the menu bar. The first ITEM statement specifies **`Go-back`** as the first selection under **`File`**. The value of the SELECTION= option corresponds to the subsequent SELECTION statement, which specifies END as the command that is issued for that selection. The second ITEM statement specifies that the SAVE command is issued for that selection.

```
      menu f;
         item 'Goback' selection=g;
         item 'Save';
         selection g 'end';
```

The ITEM statements define the selections under **`Edit`** on the menu bar.

```
      menu e;
         item 'Cancel';
         item 'Add';
```

This group of statements defines the selections under **`Scroll`** on the menu bar. If the quoted string in the ITEM statement is not a valid command, the SELECTION= option corresponds to a subsequent SELECTION statement, which specifies a valid command.

```
      menu s;
         item 'Next Obs' selection=n;
         item 'Prev Obs' selection=p;
         item 'Top';
         item 'Bottom';
         selection n 'forward';
         selection p 'backward';
```

This group of statements defines the selections under **Subset** on the menu bar. The DIALOG= option names a dialog box that is defined in a subsequent DIALOG statement.

```
menu sub;
   item 'Where' dialog=d1;
   item 'Where Clear';
```

This group of statements defines the selections under **Help** on the menu bar. The SETHELP command specifies a HELP entry that contains user-written information for this FSEDIT application. The semicolon that appears after the HELP entry name allows the HELP command to be included in the string. The HELP command invokes the HELP entry.

```
menu h;
   item 'Keys';
   item 'About this application' selection=hlp;
   selection hlp 'sethelp proclib.menucat.staffhlp.help;help';
```

WBUILD is a SAS macro. The double percent sign that precedes WBUILD is necessary to prevent PROC PMENU from expecting a field number to follow. The field numbers %1, %2, and %3 equate to the values specified by the user with the radio boxes. The field number @1 equates to the search value that the user enters. See "How the WBUILD Macro Works" on page 772.

```
 dialog d1 '%%wbuild(%1,%2,@1,%3)';
```

The TEXT statement specifies text for the dialog box that appears on line 1 and begins in column 1. The RADIOBOX statement specifies that a radio box will appear in the dialog box. DEFAULT= specifies that the first radio button (**Northeast**) will be selected by default. The RBUTTON statements specify the mutually exclusive choices for the radio buttons: **Northeast**, **Northwest**, **Southeast**, or **Southwest**. SUBSTITUTE= gives the value that is substituted for the %1 in the DIALOG statement above if that radio button is selected.

```
    text #1 @1 'Choose a region:';
    radiobox default=1;
       rbutton #3 @5 'Northeast' substitute='NE';
       rbutton #4 @5 'Northwest' substitute='NW';
       rbutton #5 @5 'Southeast' substitute='SE';
       rbutton #6 @5 'Southwest' substitute='SW';
```

The TEXT statement specifies text for the dialog box that appears on line 8 (#8) and begins in column 1 (@1). The RADIOBOX statement specifies that a radio box will appear in the dialog box. DEFAULT= specifies that the first radio button (**Pollutant A**) will be selected by default. The RBUTTON statements specify the mutually exclusive choices for the radio buttons: **Pollutant A** or **Pollutant B**. SUBSTITUTE= gives the value that is substituted for the %2 in the preceding DIALOG statement if that radio button is selected.

```
    text #8 @1 'Choose a contaminant:';
    radiobox default=1;
       rbutton #10 @5  'Pollutant A' substitute='pol_a,2';
```

```
              rbutton #11 @5  'Pollutant B' substitute='pol_b,4';
```

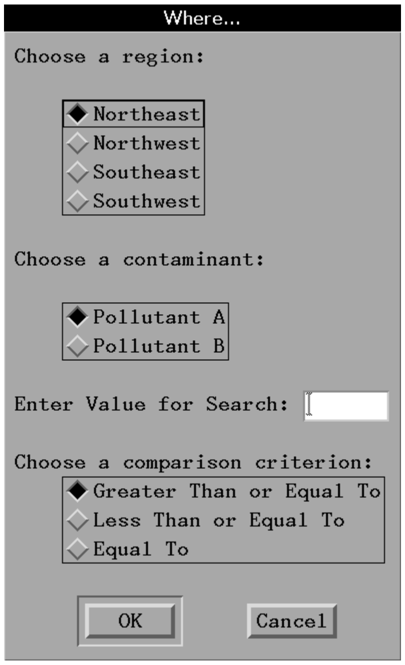
The first TEXT statement specifies text for the dialog box that appears on line 13 and begins in column 1. The second TEXT statement specifies an input field that is 6 bytes long that appears on line 13 and begins in column 25. The value that the user enters in the field is substituted for the @1 in the preceding DIALOG statement.

```
           text #13 @1 'Enter Value for Search:';
           text #13 @25 len=6;
```

The TEXT statement specifies text for the dialog box that appears on line 15 and begins in column 1. The RADIOBOX statement specifies that a radio box will appear in the dialog box. DEFAULT= specifies that the first radio button (**Greater Than or Equal To**) will be selected by default. The RBUTTON statements specify the mutually exclusive choices for the radio buttons. SUBSTITUTE= gives the value that is substituted for the %3 in the preceding DIALOG statement if that radio button is selected.

```
           text #15 @1 'Choose a comparison criterion:';
           radiobox default=1;
              rbutton #16 @5 'Greater Than or Equal To'
                             substitute='GE';
              rbutton #17 @5 'Less Than or Equal To'
                             substitute='LE';
              rbutton #18 @5 'Equal To' substitute='EQ';
quit;
```

This dialog box appears when the user selects **Subset** and then **Where**.



Associating a Menu Bar with an FSEDIT Session

The SAS data set PROCLIB.LAKES has data about several lakes. Two pollutants, pollutant A and pollutant B, were tested at each lake. Tests were conducted for pollutant A twice at each lake, and the results are recorded in the variables POL_A1 and POL_A2. Tests were conducted for pollutant B four times at each lake, and the results are recorded in the variables POL_B1 - POL_B4. Each lake is located in one of four regions. The following output lists the contents of PROCLIB.LAKES:

Output 27.1

```
                              PROCLIB.LAKES                               1

region     lake         pol_a1     pol_a2     pol_b1     pol_b2     pol_b3     pol_b4

  NE       Carr          0.24       0.99       0.95       0.36       0.44       0.67
  NE       Duraleigh     0.34       0.01       0.48       0.58       0.12       0.56
  NE       Charlie       0.40       0.48       0.29       0.56       0.52       0.95
  NE       Farmer        0.60       0.65       0.25       0.20       0.30       0.64
  NW       Canyon        0.63       0.44       0.20       0.98       0.19       0.01
  NW       Morris        0.85       0.95       0.80       0.67       0.32       0.81
  NW       Golf          0.69       0.37       0.08       0.72       0.71       0.32
  NW       Falls         0.01       0.02       0.59       0.58       0.67       0.02
  SE       Pleasant      0.16       0.96       0.71       0.35       0.35       0.48
  SE       Juliette      0.82       0.35       0.09       0.03       0.59       0.90
  SE       Massey        1.01       0.77       0.45       0.32       0.55       0.66
  SE       Delta         0.84       1.05       0.90       0.09       0.64       0.03
  SW       Alumni        0.45       0.32       0.45       0.44       0.55       0.12
  SW       New Dam       0.80       0.70       0.31       0.98       1.00       0.22
  SW       Border        0.51       0.04       0.55       0.35       0.45       0.78
  SW       Red           0.22       0.09       0.02       0.10       0.32       0.01
```

A DATA step on page 1509 creates PROCLIB.LAKES.

The following statements initiate a PROC FSEDIT session for PROCLIB.LAKES:

```
proc fsedit data=proclib.lakes screen=proclib.lakes;
run;
```

To associate the customized menu bar menu with the FSEDIT session, do any one of the following:

- enter a SETPMENU command on the command line. The command for this example is

  ```
  setpmenu proclib.menucat.project.pmenu
  ```

  Turn on the menus by entering PMENU ON on the command line.
- enter the SETPMENU command in a Command window.
- include an SCL program with the FSEDIT session that uses the customized menus and turns on the menus, for example:

  ```
  fseinit:
    call execcmd('setpmenu proclib.menucat.project.pmenu;
                  pmenu on;');
  return;
  init:
  return;
  main:
  return;
  term:
  return;
  ```

How the WBUILD Macro Works

Consider how you would learn whether any of the lakes in the Southwest region tested for a value of .50 or greater for pollutant A. Without the customized menu item, you would issue the following WHERE command in the FSEDIT window:

```
where region="SW" and (pol_a1 ge .50 or pol_a2 ge .50);
```

Using the custom menu item, you would select **Southwest**, **Pollutant A**, enter .50 as the value, and choose **Greater Than or Equal To** as the comparison criterion. Two lakes, **New Dam** and **Border**, meet the criteria.

The WBUILD macro uses the four pieces of information from the dialog box to generate a WHERE command:

- □ One of the values for region, either **NE**, **NW**, **SE**, or **SW**, becomes the value of the macro parameter REGION.
- □ Either **pol_a,2** or **pol_b,4** become the values of the PREFIX and NUMVAR macro parameters. The comma is part of the value that is passed to the WBUILD macro and serves to delimit the two parameters, PREFIX and NUMVAR.
- □ The value that the user enters for the search becomes the value of the macro parameter VALUE.
- □ The operator that the user chooses becomes the value of the macro parameter OPERATOR.

To see how the macro works, again consider the following example, in which you want to know if any of the lakes in the southwest tested for a value of .50 or greater for pollutant A. The values of the macro parameters would be

REGION	**SW**
PREFIX	**pol_a**
NUMVAR	2
VALUE	.50
OPERATOR	GE

The first %IF statement checks to make sure that the user entered a value. If a value has been entered, the macro begins to generate the WHERE command. First, the macro creates the beginning of the WHERE command:

```
where region="SW" and (
```

Next, the %DO loop executes. For pollutant A, it executes twice because NUMVAR=2. In the macro definition, the period in **&prefix.&i** concatenates **pol_a** with **1** and with **2**. At each iteration of the loop, the macro resolves PREFIX, OPERATOR, and VALUE, and it generates a part of the WHERE command. On the first iteration, it generates

```
pol_a1 GE .50
```

The %IF statement in the loop checks to see if the loop is working on its last iteration. If it is not, the macro makes a compound WHERE command by putting an **OR** between the individual clauses. The next part of the WHERE command becomes

```
OR pol_a2 GE .50
```

The loop ends after two executions for pollutant A, and the macro generates the last of the WHERE command:

```
)
```

Results from the macro are placed on the command line. The following code is the definition of the WBUILD macro. The underlined code shows the parts of the WHERE command that are text strings that the macro does not resolve:

```
%macro wbuild(region,prefix,numvar,value,operator);
       /* check to see if value is present */
    %if &value ne %then %do;
       where region="&region" AND (
               /* If the values are character,     */
               /* enclose &value in double quotes. */
         %do i=1 %to &numvar;
             &prefix.&i &operator &value
                /* if not on last variable,  */
                /* generate 'OR'             */
            %if &i ne &numvar %then %do;
                OR
            %end;
         %end;
         )
    %end;

%mend wbuild;
```

Example 4: Creating Menus for a DATA Step Window Application

Procedure features:
DIALOG statement
SELECTION statement
Other features: FILENAME statement

This example defines an application that enables the user to enter human resources data for various departments and to request reports from the data sets created by the data entry.

The first part of the example describes the PROC PMENU step that creates the menus. The subsequent sections describe how to use the menus in a DATA step window application.

Tasks include

- □ associating customized menus with a DATA step window
- □ creating menus for a DATA step window
- □ submitting SAS code from a menu selection
- □ creating a pull-down menu selection that calls a dialog box.

Program

The LIBNAME statement defines the SAS data library in which the PMENU entries are stored. The FILENAME statements define the external files in which the programs to create the windows are stored.

```
libname proclib 'SAS-data-library';
filename de     'external-file';
filename prt    'external-file';
```

CATALOG= specifies PROCLIB.MENUS as the catalog that stores menus.

```
proc pmenu catalog=proclib.menus;
```

The MENU statement specifies SELECT as the name of the catalog entry. The menus are stored in the catalog entry PROCLIB.MENUS.SELECT.PMENU.

```
   menu select;
```

The ITEM statements specify the three items on the menu bar. The value of the MENU= option is used in a subsequent MENU statement.

```
      item 'File' menu=f;
      item 'Data_Entry' menu=deptsde;
      item 'Print_Report' menu=deptsprt;
```

This group of statements defines the selections under **`File`**. The value of the SELECTION= option is used in a subsequent SELECTION statement.

```
   menu f;
      item 'End this window' selection=endwdw;
      item 'End this SAS session' selection=endsas;
      selection endwdw 'end';
      selection endsas 'bye';
```

This group of statements defines the selections under **`Data_Entry`** on the menu bar. The ITEM statements specify that **`For Dept01`** and **`For Dept02`** appear under **`Data_Entry`**. The value of the SELECTION= option equates to a subsequent SELECTION statement, which contains the string of commands that are actually submitted. The value of the DIALOG= option equates to a subsequent DIALOG statement, which describes the dialog box that appears when this item is selected.

```
   menu deptsde;
      item 'For Dept01' selection=de1;
      item 'For Dept02' selection=de2;
      item 'Other Departments' dialog=deother;
```

The commands in single quotes are submitted when the user selects **`For Dept01`** or **`For Dept02`**. The END command ends the current window and returns to the PROGRAM EDITOR window so that further commands can be submitted. The INCLUDE command includes the SAS statements that create the data entry window. The CHANGE command modifies the DATA statement in the included program so that it creates the correct data set. See “Using a Data Entry Program” on page 777. The SUBMIT command submits the DATA step program.

```
      selection de1 'end;pgm;include de;change xx 01;submit';
      selection de2 'end;pgm;include de;change xx 02;submit';
```

The DIALOG statement defines the dialog box that appears when the user selects **`Other Departments`**. The DIALOG statement modifies the command string so that the name of the department that is entered by the user is used to change **`deptxx`** in the SAS program that is included. See "Using a Data Entry Program" on page 777. The first two TEXT statements specify text that appears in the dialog box. The third TEXT statement specifies an input field. The name that is entered in this field is substituted for the @1 in the DIALOG statement.

```
      dialog deother 'end;pgm;include de;c deptxx @1;submit';
         text #1 @1 'Enter department name';
         text #2 @3 'in the form DEPT99:';
         text #2 @25 len=7;
```

This group of statements defines the choices under the **`Print_Report`** item. These ITEM statements specify that **`For Dept01`** and **`For Dept02`** appear in the pull-down menu. The value of the SELECTION= option equates to a subsequent SELECTION statement, which contains the string of commands that are actually submitted.

```
     menu deptsprt;
        item 'For Dept01' selection=prt1;
        item 'For Dept02' selection=prt2;
        item 'Other Departments' dialog=prother;
```

The commands in single quotes are submitted when the user selects **`For Dept01`** or **`For Dept02`**. The END command ends the current window and returns to the PROGRAM EDITOR window so that further commands can be submitted. The INCLUDE command includes the SAS statements that print the report. See "Printing a Program" on page 778. The CHANGE command modifies the PROC PRINT step in the included program so that it prints the correct data set. The SUBMIT command submits the PROC PRINT program.

```
        selection prt1
              'end;pgm;include prt;change xx 01 all;submit';
        selection prt2
              'end;pgm;include prt;change xx 02 all;submit';
```

The DIALOG statement defines the dialog box that appears when the user selects **`Other Departments`**. The DIALOG statement modifies the command string so that the name of the department that is entered by the user is used to change **`deptxx`** in the SAS program that is included. See "Printing a Program" on page 778. The first two TEXT statements specify text that appears in the dialog box. The third TEXT statement specifies an input field. The name entered in this field is substituted for the @1 in the DIALOG statement.

```
     dialog prother 'end;pgm;include prt;c deptxx @1 all;submit';
        text #1 @1 'Enter department name';
        text #2 @3 'in the form DEPT99:';
        text #2 @25 len=7;
```

The RUN statement ends this RUN group.

```
    run;
```

The MENU statement specifies ENTRDATA as the name of the catalog entry that this RUN group is creating. **Fi**
the menu bar. The selections available are **End this window** and **End this SAS session.**

```
      menu entrdata;
         item 'File' menu=f;
         menu f;
            item 'End this window' selection=endwdw;
            item 'End this SAS session' selection=endsas;
            selection endwdw 'end';
            selection endsas 'bye';
    run;
 quit;
```

Associating a Menu with a Window

The first group of statements defines the primary window for the application. These statements are stored in the file that is referenced by the HRWDW fileref:

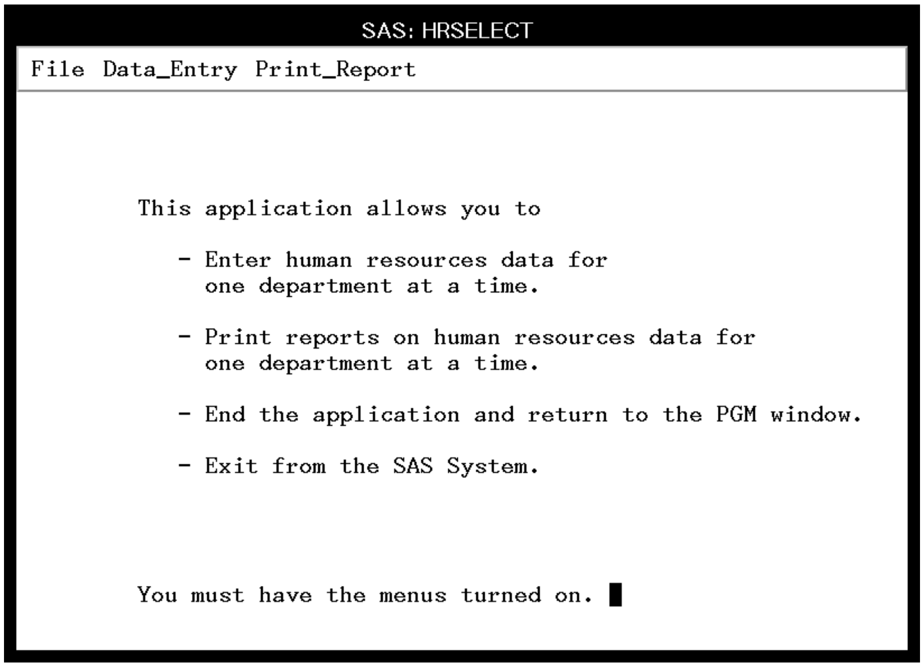
The WINDOW statement creates the HRSELECT window. MENU= associates the PROCLIB.MENUS.SELECT.PMENU entry with this window.

```
    data _null_;
       window hrselect menu=proclib.menus.select
       #4  @10 'This application allows you to'
       #6  @13 '- Enter human resources data for'
       #7  @15 'one department at a time.'
       #9  @13 '- Print reports on human resources data for'
       #10 @15 'one department at a time.'
       #12 @13 '- End the application and return to the PGM window.'
       #14 @13 '- Exit from the SAS System.'
       #19 @10 'You must have the menus turned on.';
```

The DISPLAY statement displays the window HRSELECT.

```
       display hrselect;
    run;
```

Primary window, HRSELECT.



```
This application allows you to

   - Enter human resources data for
     one department at a time.

   - Print reports on human resources data for
     one department at a time.

   - End the application and return to the PGM window.

   - Exit from the SAS System.



You must have the menus turned on.
```

Using a Data Entry Program

When the user selects **`Data_Entry`** from the menu bar in the HRSELECT window, a pull-down menu is displayed. When the user selects one of the listed departments or chooses to enter a different department, the following statements are invoked. These statements are stored in the file that is referenced by the DE fileref.

The WINDOW statement creates the HRDATA window. MENU= associates the PROCLIB.MENUS.ENTRDATA.PMENU entry with the window.

```
data proclib.deptxx;
    window hrdata menu=proclib.menus.entrdata
    #5  @10 'Employee Number'
    #8  @10 'Salary'
    #11 @10 'Employee Name'
    #5  @31 empno $4.
    #8  @31 salary 10.
    #11 @31 name $30.
    #19 @10 'Press ENTER to add the observation to the data set.';
```

The DISPLAY statement displays the HRDATA window.

```
    display hrdata;
run;
```

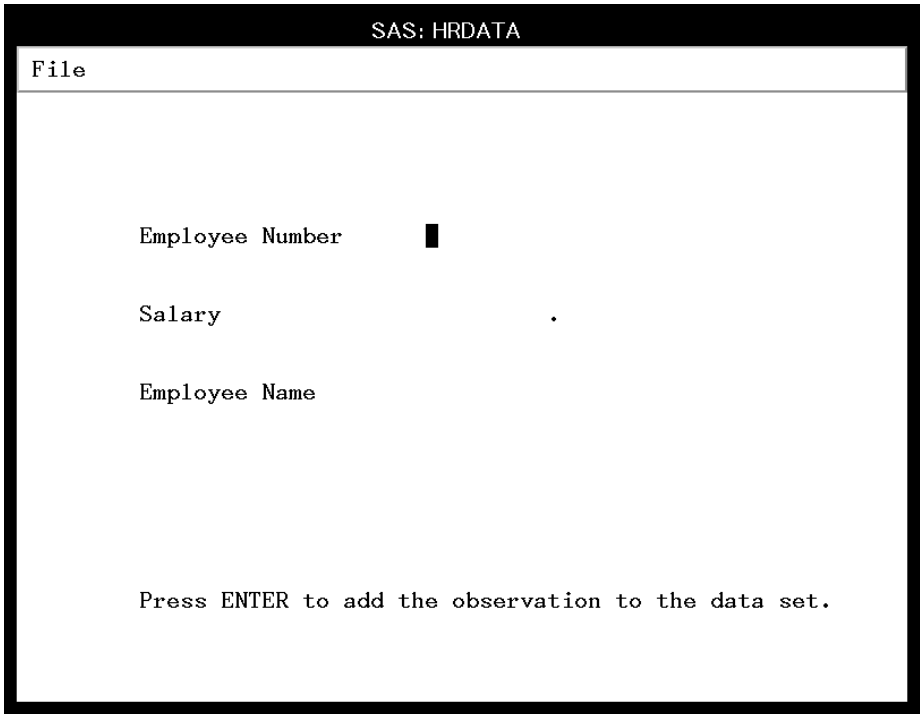
The %INCLUDE statement recalls the statements in the file HRWDW. The statements in HRWDW redisplay the primary window. See HRSELECT on page 776.

```
filename hrwdw 'external-file';
%include hrwdw;
run;
```

The SELECTION and DIALOG statements in the PROC PMENU step modify the DATA statement in this program so that the correct department name is used when the data set is created. That is, if the user selects **Other Departments** and enters **DEPT05**, the DATA statement is changed by the command string on the DIALOG statement to

```
data proclib.dept05;
```

Data entry window, HRDATA.



```
Employee Number

Salary                        .

Employee Name



Press ENTER to add the observation to the data set.
```

Printing a Program

When the user selects **Print_Report** from the menu bar, a pull-down menu is displayed. When the user selects one of the listed departments or chooses to enter a different department, the following statements are invoked. These statements are stored in the external file referenced by the PRT fileref.

PROC PRINTTO routes the output to an external file.

```
proc printto file='external-file' new;
run;
```

The **xx**'s are changed to the appropriate department number by the CHANGE command in the SELECTION or DIALOG statement in the PROC PMENU step. PROC PRINT prints that data set.

```
libname proclib 'SAS-data-library';

proc print data=proclib.deptxx;
   title 'Information for deptxx';
run;
```

This PROC PRINTTO steps restores the default output destination. See Chapter 29, “The PRINTTO Procedure,” on page 819 for documentation on PROC PRINTTO.

```
proc printto;
run;
```

The %INCLUDE statement recalls the statements in the file HRWDW. The statements in HRWDW redisplay the primary window.

```
filename hrwdw 'external-file';
%include hrwdw;
run;
```

Example 5: Associating Menus with a FRAME Application

Procedure features:
ITEM statement
MENU statement
Other features: SAS/AF software

This example creates menus for a FRAME entry and gives the steps necessary to associate the menus with a FRAME entry from SAS/AF software.

Program

```
libname proclib 'SAS-data-library';
```

CATALOG= specifies PROCLIB.MENUCAT as the catalog that stores the menus.

```
proc pmenu catalog=proclib.menucat;
```

The MENU statement specifies FRAME as the name of the catalog entry. The menus are stored in the catalog entry PROCLIB.MENUS.FRAME.PMENU.

```
   menu frame;
```

The ITEM statements specify the items in the menu bar. The value of MENU= corresponds to a subsequent MENU statement.

```
      item 'File' menu=f;
      item 'Help' menu=h;
```

The MENU statement equates to the MENU= option in a preceding ITEM statement. The ITEM statements specify the selections that are available under **`File`** in the menu bar.

```
      menu f;
         item 'Cancel';
         item 'End';
```

The MENU statement equates to the MENU= option in a preceding ITEM statement. The ITEM statements specify the selections that are available under **`Help`** on the menu bar. The value of the SELECTION= option equates to a subsequent SELECTION statement.

```
      menu h;
         item 'About the application' selection=a;
         item 'About the keys'   selection=k;
```

The SETHELP command specifies a HELP entry that contains user-written information for this application. The semicolon that appears after the HELP entry name allows the HELP command to be included in the string. The HELP command invokes the HELP entry.

```
            selection a 'sethelp proclib.menucat.app.help;help';
            selection k 'sethelp proclib.menucat.keys.help;help';
run;
quit;
```

Steps to Associate Menus with a FRAME

1 In the BUILD environment for the FRAME entry, from the menu bar, select

   View ▶ Properties Window

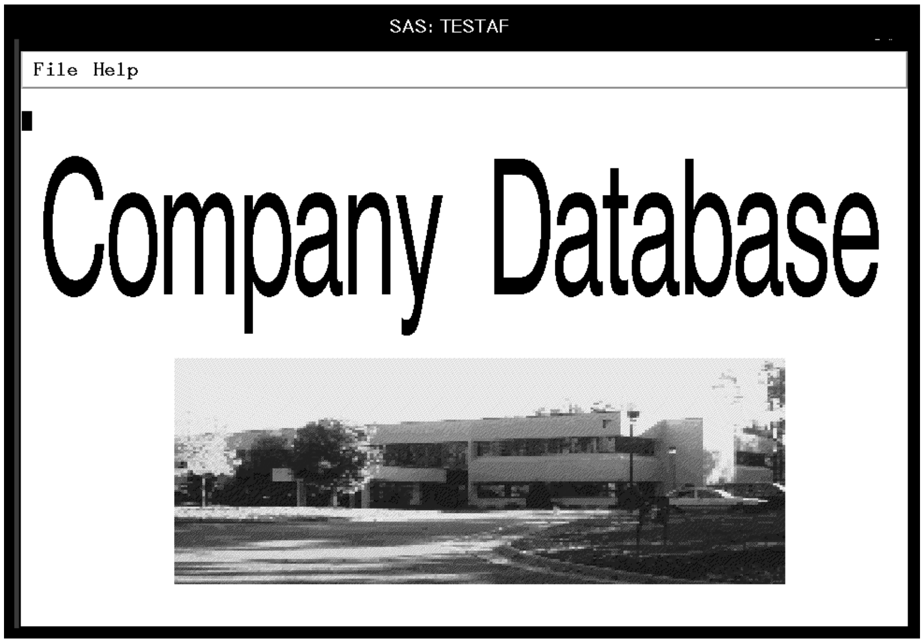
2 In the Properties window, select the **`Value`** field for the *pmenuEntry* Attribute Name. The Select An Entry window opens.

3 In the Select An Entry window, enter the name of the catalog entry that is specified in the PROC PMENU step that creates the menus.

4 Test the FRAME as follows from the menu bar of the FRAME:

   Build ▶ Test

Notice that the menus are now associated with the FRAME.



Refer to *Getting Started with the FRAME Entry: Developing Object-Oriented Applications* for more information on SAS programming with FRAME entries.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS® Procedures Guide, Version 8,* Cary, NC: SAS Institute Inc., 1999. 1729 pp.

SAS® Procedures Guide, Version 8

ISBN 1–58025–482–9

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, October 1999